**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ „FERDINAND I”**

**FACULTATEA DE COMUNICAȚII ȘI SISTEME ELECTRONICE**

**PENTRU APĂRARE ȘI SECURITATE**

**Specializarea: Comunicații pentru apărare și securitate**



**PROIECT DE DIPLOMĂ**

**Identificarea și clasarea semnalului acustic emis prin dactilografiere**

**Identification and classification of the acoustic signal emitted through typing**

CONDUCĂTOR ȘTIINȚIFIC:

**Col. Conf. Univ. Dr. Ing. Florin POPESCU**

ABSOLVENT:

**Std. Sg. Maj. Isabel-Elena Costrachevici**

Conține file

Inventariat sub nr.

Poziția din indicator:

Termen de păstrare:

Referat (1/2)

Referat (2/2)

DECLARAȚIE PE PROPRIE RĂSPUNDERE PRIVIND ORIGINALITATEA CONȚINUTULUI LUCĂRII DE LICENȚĂ

Subsemnata Costrachevici Isabel-Elena domiciliată în Mun. Piatra Neamț, str. Mihai Viteazu, nr. 27, bl. C5, sc. E, Jud. Neamț, posesor al C.I. seria NZ, nr. 107196, CNP 6020809270011, eliberat de SPCLEP Piatra Neamț, autorul lucrării de licență cu titlul „Identificarea și clasarea semnalului acustic emis prin dactilografiere”, elaborată în vederea susținerii examenului de finalizare a studiilor în sesiunea iulie 2025, declar pe propria răspundere că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Data Semnătura

Detaliere (1/2)

Detaliere (2/2)

**Abstract**

This project is aimed at developing a system capable of recognizing and classifying the acoustic signals emitted during keyboard typing with the goal of finding precise differences between the individual keystrokes based on their acoustic characteristics. The typing acoustics can potentially give away a lot of information, which is why it is important to study and classify these signals in a security and authentication context. The research was done step by step by going through successive stages, each essential to the completion of the project

The first step implies a theoretical analysis of the concepts necessary for the development of the system such as studying the acoustic signals, both from a temporal and spectral perspective, and learning about the architectures and functioning of deep learning algorithms.

The next part refers to applying the theoretical aspects for the practical implementation of the project. This step involves creating a custom dataset, which will be used to train and evaluate different architectures as well as different types of signal representations. In order to extract the acoustic features of the keystrokes, the sounds were turned into visual representations such as Melspectrograms or based on the Mel-Frequency Cepstral Coefficients or Wavelet Transform. The proposed deep learning architectures are represented by a classic Convolutional Neural Network and a modified version of the CoAtNet algorithm. Different scenarios were evaluated, in order to perform a comparative analysis and obtain the best performance.

In order to see actual results, the last step is meant to generate predictions using the trained models. The sequence of characters obtained could contain errors and to address this, Natural Language Processing techniques are applied to improve the word’s correctness. This phase is also integrated into a web interface to provide users with a clear presentation of the predictions generated by the system.

**Rezumat**

Acest proiect propune dezvoltarea unui sistem ce are ca scop recunoașterea și clasarea semnalelor emise în timpul tastării, folosind caracteristicile acustice ale acestora pentru a identifica diferențele dintre taste. Deoarece sunetele tastelor pot oferi o cantitate semnificativă de informații, este importantă analiza acestora în contextul securității și autentificării. Lucrarea a fost realizată pas cu pas prin parcurgerea unor etape succesive, fiecare esențială realizării proiectului.

Prima etapă implică o analiză teoretică a conceptelor necesare dezvoltării sistemului, precum studiul semnalelor acustice, atât din perspectiva temporală, cât și spectrală, și înțelegerea arhitecturii și funcționării algoritmilor de învățare profundă.

Următoarul pas presupune aplicarea aspectelor teoretice aprofundate în vederea implementării practice a proiectului. Această etapă presupune realizarea unui set de date propriu, pe baza căruia vor fi generate diferite tipuri de reprezentări ale semnalului, utilizate pentru antrenarea și evaluarea diferitor arhitecturi. Pentru a extrage caracteristicile acustice ale tastelor, sunetele au fost transformate în reprezentări vizuale, precum Melspectrograme sau bazate pe Coeficienții Mel-Cepstrali sau Transformata Wavelet. Arhitecturile propuse sunt reprezentate de o rețea neuronală convoluțională și o versiune modificată a algoritmului CoAtNet. Pentru a realiza o analiză comparativă, au fost testate diferite scenarii, cu scopul de a obține cele mai bune rezultate.

Pentru a putea observa un scenariu realist, ultima etapă este reprezentată de generarea predicțiilor, utilizând modelele antrenate. Secvențele de caractere obținute pot conține erori, iar pentru a soluționa acest aspect, sunt aplicate tehnici de Procesare a Limbajului Natural pentru a îmbunătăți corectitudinea cuvintelor. De asemenea, această etapă este integrată într-o interfață web pentru a oferi utilizatorilor o prezentare clară a predicțiilor generate de sistem.

**CUPRINS**

[*Listă de figuri* 13](#_Toc202355151)

[*Listă de tabele* 15](#_Toc202355152)

[*Listă de abrevieri* 17](#_Toc202355153)

[*Capitolui 1. Introducere* 19](#_Toc202355154)

[1.1 Introducere în domeniu și aria de utilizare 19](#_Toc202355155)

[1.2 Stadiul actual 21](#_Toc202355156)

[1.3 Obiectivele lucrării 23](#_Toc202355157)

[1.4 Structura lucrării 23](#_Toc202355158)

[*Capitolul 2. Noțiuni teoretice* 25](#_Toc202355159)

[2.1 Analiza semnalului acustic 25](#_Toc202355160)

[2.1.1 Semnalul acustic emis prin dactilografiere 25](#_Toc202355161)

[2.1.2 Arhitectura unei tastaturi 25](#_Toc202355162)

[2.1.3 Analiza în domeniul frecvență 26](#_Toc202355163)

[2.1.4 Coeficienții Mel-Cepstrali 29](#_Toc202355164)

[2.1.5 Transformata Wavelet 31](#_Toc202355165)

[2.2 Algoritmi de învățare profundă 32](#_Toc202355166)

[2.2.1 Rețele neuronale 33](#_Toc202355167)

[2.2.2 Rețele neuronale convoluționale 38](#_Toc202355168)

[2.2.3 CoAtNet 40](#_Toc202355169)

[2.2.4 Procesarea limbajului natural 41](#_Toc202355170)

[2.2.5 Metrici de evaluare a modelului 41](#_Toc202355171)

[2.4 Tehnologii software utilizate 44](#_Toc202355172)

[2.4.1 Python 44](#_Toc202355173)

[2.4.2 Pytorch 44](#_Toc202355174)

[2.4.3 Librosa 45](#_Toc202355175)

[2.4.4 Streamlit 45](#_Toc202355176)

[*Capitolul 3. Implementarea practică* 47](#_Toc202355177)

[3.1 Structura proiectului 47](#_Toc202355178)

[3.2 Descrierea setului de date 48](#_Toc202355179)

[3.3 Analiza semnalelor audio 49](#_Toc202355180)

[3.4 Extragerea caracteristicilor 52](#_Toc202355181)

[3.5 Arhitecturi și parametri 56](#_Toc202355182)

[3.5.1 Rețeaua neuronală convoluțională (CNN) 56](#_Toc202355183)

[3.5.2 CoAtNet 57](#_Toc202355184)

[3.6 Etapa de antrenare 59](#_Toc202355185)

[3.7 Etapa de testare 61](#_Toc202355186)

[3.8 Generarea predicțiilor 61](#_Toc202355187)

[*Capitolul 4. Rezultate experimentale* 63](#_Toc202355188)

[4.1 Performanțe CNN 63](#_Toc202355189)

[4.1.1 Antrenarea pe baza melspectrogramelor 63](#_Toc202355190)

[4.1.2 Antrenarea pe baza MFCC 66](#_Toc202355191)

[4.1.3 Antrenarea pe baza Transformatei Wavelet 68](#_Toc202355192)

[4.2 Performanțe CoAtNet 70](#_Toc202355193)

[4.2.1 Antrenarea pe baza melspectrogramelor 70](#_Toc202355194)

[4.2.2 Antrenarea pe baza MFCC 72](#_Toc202355195)

[4.2.3 Antrenarea pe baza Transformatei Wavelet 74](#_Toc202355196)

[4.3 Analiză comparativă 76](#_Toc202355197)

[4.4 Influența zgomotului 77](#_Toc202355198)

[4.5 Obținerea predicțiilor 81](#_Toc202355199)

[*Capitolul 5. Concluzii și direcții viitoare* 85](#_Toc202355200)

[*Bibliografie* 89](#_Toc202355201)

[*Anexe* 95](#_Toc202355202)

# *Listă de figuri*

[Figura 2.1 Algoritmul de extragere al coeficienților Mel-cepstrali 28](#_Toc197366077)

[Figura 2.2 Structura unui neuron artificial 32](#_Toc197366078)

[Figura 2.3 Arhitectura unei rețele neuronale 34](#_Toc197366079)

[Figura 2.4 Algoritmul gradientului descendent 36](#_Toc197366080)

[Figura 2.5 Arhitectura de bază a unei rețele neuronale convoluționale 38](#_Toc197366081)

[Figura 2.6 Matricea confuziilor 41](#_Toc197366082)

[Figura 3.1 Etapele necesare realizării proiectului 47](#_Toc197984319)

[Figura 3.2 Structura setului de date 49](#_Toc197984320)

[Figura 3.3 Forma de undă pentru tasta „q” 51](#_Toc197984321)

[Figura 3.4 Forma de undă pentru tasta „m” 51](#_Toc197984322)

[Figura 3.5 Analiza în domeniul frecvență pentru tasta „q” 52](#_Toc197984323)

[Figura 3.6 Analiza în domeniul frecvență pentru tasta „m” 52](#_Toc197984324)

[Figura 3.7 Melspectrograma pentru tasta „q” 53](#_Toc197984325)

[Figura 3.8 Melspectrograma pentru tasta „m” 53](#_Toc197984326)

[Figura 3.9 Reprezentarea MFCC a tastei „q” 54](#_Toc197984327)

[Figura 3.10 Reprezentarea MFCC a tastei „m” 54](#_Toc197984328)

[Figura 3.11 Reprezentarea Wavelet pentru tasta „q” 55](#_Toc197984329)

[Figura 3.12 Reprezentarea Wavelet pentru tasta „m” 55](#_Toc197984330)

[Figura 3.13 Structura CNN 57](#_Toc197984331)

[Figura 3.14 Structura utilizată pentru modelul CoAtNet 58](#_Toc197984332)

[Figura 4.1 Curbele de învățare - CNN, Antrenarea pe baza melspectrogramelor 65](#_Toc200542058)

[Figura 4.2 Matricea confuziilor - CNN, Antrenarea pe baza melspectrogramelor 66](#_Toc200542059)

[Figura 4.3 Metrici de evaluare - CNN, Antrenarea pe baza melspectrogramelor 67](#_Toc200542060)

[Figura 4.4 Curbele de învățare - CNN, Antrenarea pe baza MFCC 68](#_Toc200542061)

[Figura 4.5 Matricea Confuziilor - CNN, Antrenarea pe baza MFCC 68](#_Toc200542062)

[Figura 4.6 Metrici de evaluare - CNN, Antrenarea pe baza MFCC 69](#_Toc200542063)

[Figura 4.7 Curbele de învățare - CNN, Antrenarea pe baza Transformatei Wavelet 70](#_Toc200542064)

[Figura 4.8 Matricea confuziilor - CNN, Antrenarea pe baza Transformatei Wavelet 70](#_Toc200542065)

[Figura 4.9 Metrici de evaluare - CNN, Antrenarea pe baza Transformatei Wavelet 71](#_Toc200542066)

[Figura 4.10 Matricea confuziilor - CoAtNet, Antrenarea pe baza melspectrogramelor 72](#_Toc200542067)

[Figura 4.11 Curbele de învățare - CoAtNet, Antrenarea pe baza melspectrogramelor 73](#_Toc200542068)

[Figura 4.12 Metrici de evaluare - CoAtNet, Antrenarea pe baza melspectrogramelor 73](#_Toc200542069)

[Figura 4.13 Metrici de evaluare - CoAtNet, Antrenarea pe baza MFCC 75](#_Toc200542070)

[Figura 4.14 Curbele de învățare- CoAtNet, Antrenarea pe baza MFCC 75](#_Toc200542071)

[Figura 4.15 Matricea confuziilor - CoAtNet, Antrenarea pe baza MFCC 76](#_Toc200542072)

[Figura 4.16 Matricea confuziilor - CoAtNet, Antrenarea pe baza Transformatei Wavelet 77](#_Toc200542073)

[Figura 4.17 Curbele de învățare - CoAtNet, Antrenarea pe baza Transformatei Wavelet 77](#_Toc200542074)

[Figura 4.18 Metrici de evaluare - CoAtNet, Antrenarea pe baza Transformatei Wavelet 78](#_Toc200542075)

[Figura 4.19 Curbele de învățare după inserarea zgomotului 80](#_Toc200542076)

[Figura 4.20 Matricea confuziilor după inserarea zgomotului 80](#_Toc200542077)

[Figura 4.21 Metrici de evaluare după inserarea zgomotului 81](#_Toc200542078)

[Figura 4.22 Curbele de învățare după inserarea zgomotului în cadrul tutror etapelor 82](#_Toc200542079)

[Figura 4.23 Matricea confuziilor după inserarea zgomotului în cadrul tuturor etapelor 82](#_Toc200542080)

[Figura 4.24 Metrici de evaluare după inserarea zgomotului în cadrul tuturor etapelor 83](#_Toc200542081)

[Figura 4.25 Realizarea predicțiilor - Cazul I 84](#_Toc200542082)

[Figura 4.26 Realizarea predicțiilor - Cazul II 85](#_Toc200542083)

# *Listă de tabele*

[Tabel 1 Configurația arhitecturală a modelului CoAtNet-0 58](#_Toc197984335)

[Tabel 2 Parametrii de antrenare 61](#_Toc197984336)

# *Listă de abrevieri*

ABS Acrylonitrile Butadiene Styrene

AI Artificial Intelligence

ANN Artificial Neural Network

CNN Convolutional Neural Network

CWT Continuous Wavelet Transform

DCT Discrete Cosine Transform

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

GPU Graphics Processing Unit

HMM Hidden Markov Model

MFCC Mel-Frequency Cepstral Coefficients

ML Machine Learning

NLP Natural Language Processing

PBS Polybutylene Terephthalate

PCB Printed Circuit Board

PIL Python Imaging Library

ReLU Rectified Linear Unit

SGD Stochastic Gradient Descent

STFT Short-Time Fourier Transform

SVM Support Vector Machines

# *Capitolui 1. Introducere*

## 1.1 Introducere în domeniu și aria de utilizare

Dactilografierea poate fi descrisă ca procesul de scriere a unui text prin utilizarea unei tastaturi. Fiecare tastă produce un sunet unic, iar analizând aceste semnale acustice este posibilă determinarea tastelor care au fost apăsate. Sunetele distincte se datorează faptului că tastele sunt poziționate diferit pe tastatură, chiar dacă studiile indică și existența unor posibili factori secundari, precum influența tastelor alăturate sau diferențele prezente în construcția tastelor, aceștia sunt considerați neglijabili [1].

Semnale acustice obținute în urma dactilografierii pot fi utilizate în cadrul atacurilor de tip canal lateral, folosind tehnici de procesare a semnalelor și de recunoaștere a tiparelor. Un atac de tip canal lateral este un atac cibernetic care se bazează pe exploatarea implementării fizice în vederea obținerea informațiilor. În cadrul acestor atacuri, există mai multe aspecte ce pot fi utilizate pentru a obține informații cum ar fi: consumul de energie, timpul de execuție, emisiile electromagnetice sau emisiile acustice [2].

Procesarea semnalelor acustice se referă la analiza acestora pentru a extrage caracteristicile relevante, ceea ce presupune o serie de etape, precum captarea sunetului cu ajutorul unui microfon, preprocesarea, ultima etapă fiind extragerea caracteristicilor. Datele devin mai ușor de gestionat prin aplicarea procesului de extragere a trăsăturilor, deoarece acesta elimină toate caracteristicile mai puțin relevante, fără a pierde informații esențiale [3]. Ulterior, sunt utilizate diferite metode de recunoaștere a tiparelor pentru identificarea tastelor care implică fie analiza temporală [4], fie cea spectrală. Analiza în frecvență se împarte astfel: analiză statistică [5], [6] și folosirea conceptelor de inteligență artificială, mai exact algoritmii de învățare automată, respectiv învățare profundă. [1], [2], [6], [7], [8], [9], [10]. Această lucrare se concentrează pe aplicarea tehnicilor de învățare profundă.

Sociologul Sherry Turkle, care își axează cercetarea asupra dinamicii relației dintre om și tehnologie, afirmă că „inteligența artificială este adesea descrisă ca arta și știința de a face mașinile să realizeze lucruri care ar fi considerate inteligente dacă ar fi făcute de oameni.”[11]. Deși definiția clasică caracterizează inteligența artificială ca pe o emulație a inteligenței umane, aceasta poate depăși această limitare. Dacă sunt analizate procesele prin care AI ajunge la obținerea rezultatelor, mai exact asupra mecanismelor de funcționare și a structurilor utilizate, se poate observa o perspectivă diferită asupra naturii inteligenței artificiale.

Inteligența artificială reprezintă una dintre cele mai importante tehnologii actuale și are o influență majoră asupra mai multor domenii cum ar fi: finanțele, sănătatea, transporturile, justiția etc. În domeniul financiar, aceasta permite automatizarea deciziilor în legătură cu aprobarea împrumuturilor sau realizarea unor investiții personalizate. În sistemul sanitar, inteligența artificială reușește să îmbunătățească acuratețea diagnosticelor, în special în imagistica medicală și în gestionarea bolilor cronice. Sistemul de justiție utilizează tehnologii bazate pe AI precum recunoaștere facială, un concept tot mai răspândit, asemenea analizei amprentelor. În transporturi, AI este integrată în vehicule autonome prin intermediul senzorilor și a algoritmilor de învățare profundă care permit vehiculelor navigarea independentă [12].

Însă pentru a înțelege amploarea transformărilor pe care inteligența artificială le generează în cadrul societății, este important să ne concentrăm asupra evoluției acestui domeniu. Conceptul de inteligență artificială a apărut în timpul celui de-al Doilea Război Mondial, ca urmare a necesității de a dezvolta mașini și tehnologii care să poată rezolva probleme complexe. Învățarea automată este un subdomeniu al inteligenței artificiale care are ca scop dezvoltarea algoritmilor care permit computerelor să învețe și să ia decizii cu ajutorul datelor, fără a fi programate explicit. Câteva subiecte abordate prin prisma conceptului ML sunt: recunoașterea imaginilor și a tiparelor, procesarea limbajului natural, analiza sentimentelor, analiza comportamentului etc. [13]. Progresele hardware și software au condus la dezvoltarea conceptului de învățare profundă, un subdomeniu specializat al învățării automate, lucru care indică interdependența dintre aceste tehnologii. Modul de funcționare al acestor algoritmi este bazat pe conceptele fundamentale ale creierului uman, precum neuroni, interconectarea acestora, modul de procesare ierarhic al informațiilor etc. Acești algoritmi pot învăța reprezentări complexe ale datelor și pot recunoaște tipare subtile, ceea ce îi face potriviți pentru sarcina proiectului.

Tema studiată în cadrul lucrării este „Identificarea și clasarea semnalului acustic emis prin dactilografiere”. Motivația dezvoltării în direcția acestei teme este importanța evidențierii vulnerabilităților sistemelor datorate emisiilor generate de implementarea fizică, în special emisia semnalelor acustice.

În 2004, R. Agrawal a demonstrat, pentru prima oară, potențialul exploatării emisiilor acustice ale tastaturii pentru extragerea datelor rezultate in urma tastării [1]. De atunci, numeroase lucrări au fost realizate pentru a îmbunătăți performanțele studiilor anterioare prin diferite metode și în diverse scenarii.

## 1.2 Stadiul actual

Există mai multe maniere de abordare a temei, fiecare oferind soluții diferite pentru tratarea subiectului.

În publicația [1], menționată anterior, este propusă o metodă care folosește Transformata Fourier aplicată segmentelor audio, pe baza căreia este antrenată o rețea neuronală pentru a recunoaște apăsările tastelor. Sunt prezentate două scenarii principale: distingerea dintre două taste, unde s-a obținut un grad ridicat de acuratețe și distingerea dintre 30 de taste, unde acuratețea maximă atinsă a fost de 79%. De asemenea, studiul explorează și situația în care rețeaua este antrenată cu ajutorul unei tastaturi și testată cu ajutorul altei tastaturi, caz în care tasta corectă s-a aflat în primele 4 predicții în 52% dintre teste. Ultimul experiment a constat în studierea impactului stilurilor diferite de dactilografiere. S-a observat că, atunci când rețeaua este antrenată folosind un set de date care include apăsări de taste efectuate în stiluri diferite, tastele sunt recunoscute cu o acuratețe similară cazurilor inițiale.

Lucrarea [4] continuă cercetările și implementează o metodă alternativă bazată pe învățare nesupervizată, care nu necesită date de antrenament etichetate. Atacul propus utilizează modele statistice de limbaj, coeficienți Mel-cepstrali (MFCC) și modele Markov (HMM) pentru identificarea și clasarea semnalelor acustice. Studiul demonstrează faptul că folosirea caracteristicilor cepstrale duce la rate de recunoaștere a tastelor superioare în comparație cu metoda Transformata Fourier Rapidă (FFT). Ulterior, a fost introdus un mecanism de învățare supervizată bazat pe feedback, care utilizează rezultatele corecte ca date de antrenament pentru îmbunătățirea performanțelor. Acuratețea maximă obținută în urma procesului de invățăre nesupervizată este 87,195% pentru caractere și 74,57% pentru cuvinte și respectiv, în urma introducerii mecanismului bazat pe feedback 96,34% pentru caractere și 90,46% pentru cuvinte.

Ideile prezentate în [5] oferă o perspectivă nouă asupra temei, implementând o metodă care nu necesită antrenament, capabilă să recunoască cuvinte formate din 7-13 litere. Metoda implică segmentarea semnalului în evenimente de apăsare și eliberare a tastei, măsurarea similarității acustice, utilizând corelația și construirea unei matrice de similaritate. Pe baza acesteia se realizează constrângeri, folosite ulterior pentru clasificarea tastelor. Rata generală de succes este de 73%, fiind direct afectată de lungimea cuvintelor.

Publicația [10] demonstrează un atac care exploatează spectrogramele realizate pe baza sunetelor tastelor și o rețea neuronală simplă formată doar din două straturi. Apăsările tastelor sunt procesate folosind Transformata Fourier pe Termen Scurt (STFT) pentru a crea spectrogramele, valorile frecvențelor fiind introduse în vectori de caracteristici, utilizați pentru antrenarea rețelei. Acuratețea rezultată este 72,3% în experimentele cu 80 de exemple pentru fiecare tastă și a crescut până la 98.8% atunci când experimentele au fost realizate cu un set de date considerabil mai mic.

În lucrarea [2], Mubarek Mohammed explorează mai mulți algoritmi de învățare automată. Studiul formulează două sarcini de clasificare: distingerea tastelor de tip spațiu (space, enter) de litere și identificarea literelor individual. Apăsările tastelor sunt convertite în Melspectrograme, iar pentru extragerea trăsăturilor este folosită rețeaua neuronală pre-antrenată ResNet18. Aceste trăsături sunt ulterior clasate cu ajutorul algoritmilor precum Support Vector Machines (SVM), Random Forest sau XGBoost. Modelul SVM a obținut cele mai bune rezultate, cu o acuratețe de 96% în diferențierea literelor față de tastele de tip spațiu și o acuratețe de 53% în recunoașterea literelor.

Lucrarea [7] introduce o metodă capabilă să recunoască tastele, utilizând algoritmul de învățare profundă CoAtNet, un model care combină straturi convoluționale cu straturi de „self-attention”[[1]](#footnote-1). Pentru extragerea caracteristicilor acustice, sunt utilizate Melspectrogramele, deoarece au o structură vizuală compatibilă cu modelul ales în cadrul lucrării. Acuratețea obținută în cazul înregistrărilor realizate cu un telefon este 95%, iar în cazul înregistrărilor efectuate prin intermediul platformei Zoom acuratețea este 93%. Studiul arată că mai multe perspective ale atacului obțin rezultate bune, atât local cu ajutorul unui microfon, cât și remote, folosind platforme de comunicare online.

Un studiu recent [14] propune o abordare, ce constă în construirea unui set de date propriu ce conține 36 caractere alfanumerice și folosirea coeficienților Mel-cepstrali (MFCC) pentru extragerea trăsăturilor audio. În etapa de antrenare, este implementată o rețea neuronală convoluțională (CNN) compusă din straturi de convoluție, normalizare, grupare, dezactivare a neuronilor și straturi dense. Sistemul obține o acuratețe de 94,6% în recunoașterea caracterelor, demonstrând eficiența metodei propuse.

Toate aceste studii evidențiază atât pericolul, cât și practicalitatea atacurilor bazate pe semnalele acustice emise în timpul tastării, demonstrând că de-a lungul anilor, a existat o evoluție clară în ceea ce privește performanțele sistemelor dezvoltate. Cu toate acestea, potențialul dezvoltării rămâne considerabil, având în vedere progresele tehnologice din domeniu, fapt ce lasă deschisă oportunitatea de cercetare în această direcție.

## 1.3 Obiectivele lucrării

Lucrarea are ca obiectiv principal implementarea unui sistem de identificare și clasare a semnalului acustic emis prin dactilografiere. Pentru atingerea acestui scop au fost parcurși mai mulți pași astfel:

Studiul semnalului acustic emis de taste, în special caracteristicile definitorii și analiza acestuia în vederea extragerii trăsăturilor audio necesare în etapa următoare a proiectului. Se definesc la nivel teoretic caracteristicile semnalului, iar ulterior se implementează la nivel practic mai multe tehnici specifice de procesare audio în vederea extragerii acestor caracteristici.

Înțelegerea funcționării algoritmilor de învățare profundă potriviți utilizării în cadrul proiectului și implementarea practică a acestora. Este prezentată funcționarea și componentele unei rețele neuronale, utilizată frecvent în procesarea datelor audio. În cadrul proiectului sunt propuse următoarele arhitecturi: o rețea neuronală convoluțională și o versiune modificată a algoritmului CoAtNet. Etapele de antrenare și testare vor fi implementate pe baza unui set de date propriu, iar parametrii utilizați în timpul antrenării vor rămâne aceeași, indiferent de arhitectura utilizată. Rezultatele finale obținute vor fi comparate pentru a observa performanțele fiecărei metode aplicate. Comparația se va concentra atât pe tipul de caracteristici utilizate, cât și pe algoritmul de clasificare pentru a determina ce scenariu oferă cea mai mare acuratețe în recunoașterea tastelor.

Generarea predicțiilor și corectarea acestora prin aplicarea tehnicilor de Procesare a Limbajului Natural.Se detaliază cum, pe baza modelelor antrenate, sunt generate predicții prin clasificarea semnalelor audio și atribuirea unei taste. Însă, în cadrul acestui proces, pot aparea erori, astfel ducând la apariția unor predicții false, iar pentru a îmbunătăți acest lucru, sunt utilizate tehnici de Procesare a Limbajului Natural, care oferă posibile alternative pentru înlocuirea cuvintelor greșite.

## 1.4 Structura lucrării

Lucrarea este structurată în 5 capitole principale, fiecare prezentând aspecte esențiale pentru realizarea proiectului.

Primul capitol oferă o perspectivă generală a temei, mai exact: introducerea în domeniul studiat și aria de utilizare, prezentarea stadiului actual și obiectivele propuse.

Capitolul 2 aduce în atenție aspectele teoretice necesare înțelegerii conceptelor adoptate în cadrul proiectului, cum ar fi: algoritmii de prelucrare a semnalului acustic și arhitecturile implementate în contextul sistemului.

Capitolul 3 ilustrează implementarea practică, bazată pe principiile prezentate în capitolul 2. Implementarea poate fi împărțită în mai multe etape: realizarea setului de date, procesarea acestuia, implementarea arhitecturilor, atât cât pentru etapa de antrenare, cât și pentru etapa de testare și generarea predicțiilor.

Capitolul 4 furnizează informații despre rezultatele și performanțele obținute de metodele propuse, precum și comparații pe baza acestora.

Ultimul capitol expune concluziile obținute în urma realizării proiectului și, în plus, sunt prezentate dificultățile întâmpinate și sunt propuse direcții de cercetare viitoare pentru a extinde rezultatele obținute.

# *Capitolul 2. Noțiuni teoretice*

Acest capitol prezintă noțiunile teoretice fundamentale necesare implementării practice a proiectului conform cerinței, începând cu partea de analiză a semnalului și de caracterizare a trăsăturilor extrase și continuând cu partea de învățare profundă, realizând o descriere a funcționării arhitecturilor implementate.

## 2.1 Analiza semnalului acustic

### 2.1.1 Semnalul acustic emis prin dactilografiere

Unda acustică este o oscilație mecanică a presiunii care se propagă printr-un mediu (solid, lichiz, gaz sau plasmă) sub forma unui model periodic, transmițând energie de la un punct la altul. Unda acustică transmite sunetul prin stimularea organelor auditive care produc astfel senzația de auz, motiv pentru care este denumită și semnal acustic [15].

Semnalele pot fi clasificate în funcție de mai multe criterii. În ceea ce privește continuitatea, ele se împart în semnale continue și discrete. Din perspectiva comportamentului în timp, există semnale periodice și neperiodice, iar în funcție de predictibilate, acestea pot fi deterministe sau aleatoare. Semnalele acustice emise prin dactilografiere sunt considerate semnale tranzitorii, o subcategorie a semnalelor neperiodice, fiind caracterizate de o durată scurtă, de ordinul câtorva milisecunde [16].

Semnalele acustice emise prin apăsarea tastelor pot fi caracterizate prin două regiuni distincte: de apăsare a tastei și respectiv de eliberare. Studiile arată că în funcție de forța cu care este apăsată tasta, atât regiunea de apăsare, cât și cea de eliberare prezintă, în reprezentarea în domeniul timp, amplitudini mai ridicate care ulterior scad. [6] În domeniul frecvenței, sunetele tastelor sunt concentrate în intervalul jos-mediu al frecvențelor, acest lucru fiind influențat de factori precum stilul dactilografierii sau poziția în care este apăsată tasta [7].

### 2.1.2 Arhitectura unei tastaturi

Componentele unei tastaturi lucrează împreună pentru a converti apăsări ale tastelor în semnale electrice pentru a fi interpretate de un computer. Fiecare element are un rol important și poate influența direct performanța utilizării tastaturii. În contextul semnalelor acustice, componentele care pot influența caracteristicile sunetului sunt capacele tastelor și comutatoarele.

Capacele tastelor reprezintă componenta situată deasupra comutatoarelor și care intră în contact direct cu degetele utilizatorului. Ele pot influența sunetul emis la apăsare în funcție de materialul din care sunt realizate. Cele mai comune materiale sunt PBT sau ABS, dar există totuși posibilitatea utilizării materialelor precum rășină, lemn sau aluminiu. De asemenea, capacele pot avea profiluri diferite și se disting prin variații de înălțime, formă și geometria rândurilor de taste [17].

Comutatoarele tastaturilor prezintă mai multe caracteristici: forța necesară pentru apăsare, senzația apăsării, distanța de cursă a tastei, tipul de iluminare etc. Există trei tipuri de comutatoare, fiecare oferind o modalitate diferită a tastării: clicky (feedback tactil insoțit de un sunet puternic), tactil (feedback tactil însoțit de un sunet atenuat) și liniar (apăsare mai silențioasă). Există și tastaturi, adesea întâlnite pe laptop, care folosesc comutatoare din cauciuc ce nu oferă feedback tactil și sunt silențioase [17].

De asemena, tastele de dimensiuni mai mari, cum ar fi Space sau Enter, dispun de un mecanism diferit de funcționare care include stabilizatoare ce au scopul adaptării la suprafața extinsă. Aceste stabilizatoare conduc la generarea unui sunet distinct la apăsare în comparație cu tastele standard [17].

### 2.1.3 Analiza în domeniul frecvență

Majoritatea algoritmilor de învățare profundă utilizează caracteristicile obținute în urma analizei în domeniul frecvență, iar acest principiu se aplică și în cadrul proiectului de față. În domeniul timp, semnalul este reprezentat de o formă de undă ce ilustrează cum variază amplitudinea în timp. Reprezentările acestor semnale acustice în domeniul timp nu evidențiază diferențele dintre taste suficient, ceea ce le face mai puțin adecvate pentru antrenarea unor algoritmi de învațare automată. În schimb, reprezentările în domeniul frecvență ilustrează mai bine semnalele și evidențiază variații între taste, contribuind astfel la performanțe mai bune ale modelelor [10].

Cea mai utilizată tehnică prin care se realizează trecerea unui semnal din perspectiva temporală în cea spectrală este Transformata Fourier.

Transformata Fourier este o tehnică matematică fundamentală utilizată pentru analiza conținutului spectral al semnalelor. Există două tipuri Transformata Fourier Continuă, care se aplică semnalelor analogice, și Transformata Fourier Discretă (DFT), care este utilizată în cazul semnalelor digitale, adică eșantionate și cuantificate. În domeniul procesării audio, este adesea folosită Transformata Fourier Rapidă (FFT), o metodă eficientă pentru calcularea DFT, care descompune semnalul reprezentat în timp în componente spectrale, ilustrând astfel amplitudinea și faza fiecărei componente [18].

În primul rând, este important de menționat că semnalul acustic emis în timpul dactilografierii are o natură continuă. Pentru a putea fi procesat digital, acesta trebuie convertit într-un semnal discret, fapt ce implică eșantionarea undei sonore la intervale regulate de timp. Rata de eșantionare, exprimată în Hz, reprezintă numărul de eșantioane într-o secundă și determină rezoluția semnalului digital. O rată de eșantionare mai mare permite captarea mai precisă a detaliilor acustice [18].

(2.1)

Ecuația (2.1) translatează o secvență de *N* numere complexe *x[n]* din domeniul timp într-o secvență de *N* numere complexe *X[k]* în domeniul frecvență. FFT reprezintă o metodă optimizată care oferă aceleași rezultate ca DFT, dar care necesită mult mai puține operații de calcul. Această creștere a eficienței se datorează descompunerii recursive a Transformatei Fourier Discrete în transformate de dimensiuni mai mici [18].

Dimensiunea unui cadru se referă la numărul de eșantioane folosite în fiecare operație FFT și influențează semnificativ echilibrul dintre rezoluția în frecvență (superioară în cazul cadrelor mai mari) și rezoluția în timp (mai bună în cazul cadrelor mai mici) [18]. Din cauza procesului de fragmentare a semnalului, pot apărea discontinuități la capetele cadrelor, care pot duce la distorsiuni. Pentru atenua acest efect, se aplică ferestre care au rolul de a netezi marginile fiecărui cadru, reducând astfel variațiile bruște. Fereastra Hann este adesea întâlnită în procesarea semnalelor, în special analiza audio, iar prin aplicarea sa pe fiecare cadru, aceasta contribuie la obținerea unor reprezentări spectrale clare și precise [18], [19].

În practică, algoritmul FFT impune ca dimensiunea ferestrei să fie o putere a numărului 2 pentru a optimiza viteza de procesare. După segmentarea semnalului în cadre, se aplică FFT fiecărui cadru în parte, generând o succesiune de reprezentări spectrale locale, care reflectă evoluția conținutului în frecvență a semnalului de-a lungul timpului. [18]

Semnalul acustic emis prin apăsarea tastelor este un semnal nestaționar, caracterizat prin frecvențe și amplitudini care variază în timp. Spre deosebire de semnalele simple, cum ar fi cele sinusoidale sau dreptunghiulare, semnalele nestaționare nu urmează tipare clare, ceea ce le face mai dificil de analizat. Cele mai eficiente metode de analiză spectrală în acest context sunt STFT sau Transformata Wavelet, ambele capabile să ilustreze variațiile spectrale ale semnalului în funcție de timp. [20]

Transformata Fourier pe Termen Scurt reprezintă o versiune modificată a Transformatei Fourier, concepută pentru a analiza conținutul în frecvență al unui semnal în intervale scurte de timp. Aceasta presupune segmentarea semnalului în porțiuni suprapuse și aplicarea FFT fiecărei porțiuni individuale. STFT este potrivită pentru analiza semnalelor nestaționare, deoarece presupune că, pe intervale mici, un semnal nestaționar se comportă ca un semnal staționar. Principala limitare a STFT este utilizarea ferestrelor de lungime fixă ceea ce înseamnă rezoluție constantă atât în timp, cât și în frecvență. [20]

Rezultatul aplicării STFT este o reprezentare timp-frecvență cunoscută sub numele de spectrogramă. Spectrograma reprezintă imaginea fundamentală a spectrului de frecvențe, evidențiind modul în care componentele de frecvență fluctuează în timp. Pe axa absiceselor este reprezentat timpul, pe axa ordonatelor este reprezentată frecvența, iar intensitatea culorilor indică amplitudinea fiecărei componentă spectrală la momente specifice. De obicei, axa frecvenței este reprezentată pe o scară logaritmică, iar valorile amplitudinii sunt exprimate în decibeli. [21]

Scara Mel reprezintă o transformare logaritmică a frecvențelor semnalelor, concepută pentru a reflecta modul în care urechea umană percepe sunetele. Principiul fundamental este ca distanțele egale pe scara Mel să corespundă percepției umane a distanțelor egale între sunete. Studiile arată că oamenii sunt mai buni la detectarea frecvențelor joase în comparație cu cele înalte. De exemplu, este mai ușor să detectăm diferența dintre 100 și 200 Hz, decât dintre 10100 și 10200 Hz, cu toate că distanța numerică este acceași. [22]

(2.2)

Formula (2.2) descrie conversia frecvențelor din scara Hz în scara Mel. În această ecuație, logaritmul utilizat se referă la cel natural (logaritm în baza e). Dacă s-ar fi folosit logaritm în baza 10, atunci coeficientul din formulă ar fi avut o valoarea diferită. [22]

Melspectrogramele reprezintă o versiune adaptată a spectrogramelor clasice, în care frecvențele sunt ilustrate pe scara Mel. Această transformare comprimă informația spectrală, oferind o reprezentare mai relevantă din punct de vedere perceptiv. Datorită acestei caracteristici, melspectrogramele sunt adesea utilizate în sistemele de DL, deoarece oferă informații compacte, reducând dimensiunea datelor de intrare și generează reprezentări semnificative din care algoritmii pot învăța ușor. [22]

În plus, pornind de la operația fundamentală care permite trecerea din domeniul timp în domeniul frecvență, pot fi aplicate prelucrări suplimentare pentru a extrage și alte trăsături ale semnalului acustic.

### 2.1.4 Coeficienții Mel-Cepstrali

Deși coeficienții Mel-cepstrali sunt concepuți pentru a prioritiza frecvențele relevante în vorbirea umană, aceștia au fost utilizați în numeroase studii pentru extragerea trăsăturilor acustice în contextul recunoașterii tastelor [4], [9], [14].

Coeficienții Mel-cepstrali sunt derivați din melspectrograme, dar implică câteva etape de procesare suplimentare, deoarece aceștia reprezintă o reprezentare mai compactă a trăsăturilor spectrale. MFCC sunt calculați urmând 5 pași succesivi: pre-accentuarea semnalului pentru a amplifica frecvențele înalte, segmentarea și aplicarea unei ferestre, aplicarea Transformatei Fourier Rapidă, filtrarea cu ajutorul scării Mel și, în final, aplicarea Transformatei Cosinus Discret (DCT) [3].

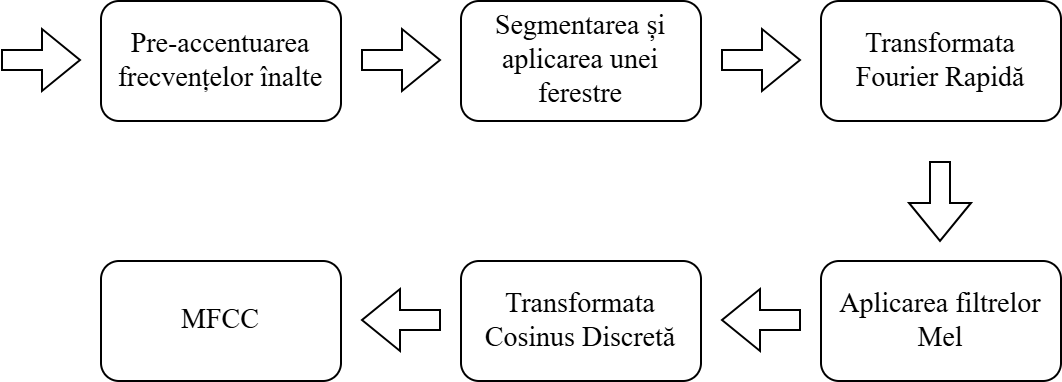


Figura 2.1 Algoritmul de extragere al coeficienților Mel-cepstrali

Pre-accentuarea este un pas important în compensarea pierderilor din componentele de frecvență înaltă ale semnalului care pot apărea în timpul generării semnalului. Această operație constă în aplicarea unui filtru trece-sus, care are scopul de a amplifica frecvențele înalte, adică acele componente care se schimbă rapid în timp [23].

Ulterior, semnalul este împărțit în mai multe cadre pentru a aplica FFT. Dimensiunea cadrelor poate varia între 25 ms și până a 500 ms. De asemenea, un parametru important este cel care indică numărul de eșantioane dintre începuturile cadrelor consecutive. Dacă acesta egal cu dimensiunea unui cadru, atunci nu va exista suprapunere între cadre. Această suprapunere a cadrelor poate ajuta la reducerea pierderilor de informație de la marginea cadrelor, astfel contribuind la o reprezentare mai continua a caracteristicilor. Urmează aplicarea unei ferestre Hann asupra fiecărui cadru și ulterior aplicarea Transformatei Fourier Rapide [3], [23].

Tranziția către domeniul spectral a fost prima etapă importantă în obținerea coeficienților Mel-cepstrali. Următoarea etapă constă în aplicarea unui set de filtre trece-bandă, cunoscute sub denumirea de filtre Mel. Aceste filtre au fost inițial dezvoltate pentru analiza vorbirii și, asemenea modului în care urechea umană percepe sunetele, acestea au ca scop extragerea unei reprezentări neliniare a semnalului. Filtrele implementate au formă triunghiulară, iar scopul lor este de a obține valorile corespunzătoare frecvențelor centrale, întrucât fiecare filtru atinge valoarea 1 în dreptul frecvenței centrale și descrește către 0, aceasta fiind frecvența centrală dintre două filtre alăturate. Procesul de prelucrare implică înmulțirea spectrului semnalului cu fiecare dintre filtrele Mel, continuată de transformarea pe o scară logaritmică și obținerea coeficienților. Funcția de transfer a unui filtru poate fi calculată astfel: [24] [3]

(2.3)

unde, este frecvența centrală a filtrului și .

Ultimul pas înainte de obținerea coeficienților Mel-cepstrali este aplicarea Transformatei Cosinus Discretă. DCT exprimă o secvență finită de date ca o sumă de funcții cosinus care oscilează la diferite frecvențe. Mai exact, readuce valorile în domeniul frecvență În procesul obținerii MFCC, DCT se aplică bancului de filtre Mel pentru a selecta cei mai importanți coeficienți. DCT se calculează astfel: [25]

(2.4)

Semnalul este transformat într-o reprezentare cepstrală, rezultatul fiind un set de coeficienți care alcătuiesc o imagine spectrală compactă a semnalului. În literatura de specialitate, cele mai multe lucrări propun metoda standard de calcul a MFCC, descrisă anterior, folosind 13 sau mai puțini coeficienți, considerați suficienți pentru a capta caracteristicile semnalului [3].

### 2.1.5 Transformata Wavelet

Așa cum a fost menționat anterior, semnalele acustice emise prin dactilografiere sunt semnale nestaționare, și din acest motiv, o metodă eficientă de analiză a acestora este reprezentată de Transformata Wavelet.

Un dezavantaj major al transformatei Fourier este că oferă o reprezentare spectrală globală, evidențiind frecvențele care persistă pe întreaga durată a semnalului, fără a furniza informații despre momentul apariției acestora. Transformata Wavelet permite o analiză simultană în timp și în frecvență, oferind o reprezentare mai detaliată. Deși STFT reprezintă o alternativă viabilă pentru analiza semnalelor nestaționare, Transformata Wavelet este adesea preferată, deoarece STFT implică segmentarea semnalului iar apoi aplicarea FFT asupra fiecărui segment, limitând astfel flexibilitatea rezoluției [26].

Transformata Wavelet reprezintă o tehnică matematică folosită pentru a descompune un semnal în componente obținute prin scalarea și translatarea unei funcții denumită wavelet. Transformata Wavelet Continuă (CWT) a unui semnal se definește ca: [27]

(2.5)

În ecuația (2.5), *ψ(t)* reprezintă funcția wavelet, aleasă în funcție de caracteristicile semnalului analizat. Cele mai utilizate exemple sunt: Morlet, folosită pentru semnale oscilatorii, Mexican Hat, ideală pentru evenimente tranzitorii, și Haar, potrivită pentru detecția marginilor. Parametrul *a* reprezintă factorul de scalare, pentru care valorile mari corespund frecvențelor joase, iar valorile mici evidențiază frecvențele înalte. Parametrul *b* este factorul de translatare, care caracterizează poziționarea funcției în timp: scăderea valorii o deplasează spre stânga, iar creșterea o deplasează spre dreapta. Ideea de bază a funcționării constă în determinarea gradului de corespondență dintre semnalul analizat și o funcție wavelet. Acest lucru se realizează prin convoluția semnalul cu un set de funcții wavelet, fiecare având parametri specifici. Rezultatul operației de convoluție ne oferă un coeficient, care indică cât de bine se potrivește waveletul cu semnalul la o anumită scară într-un anumit moment de timp [26], [27].

Rezultatul obținut în urma aplicării Transformatei Wavelet este o reprezentare timp-frecvență a semnalului, ce ne arată cum evoluează componentele spectrale în timp [27].

Funcția wavelet Morlet este o componentă fundamentală a analizei timp-frecvență, inițial concepută pentru procesarea semnalelor seismice. Este definită ca o undă sinusoidală modulată printr-o anvelopă Gaussiană[[2]](#footnote-2). În cadrul CWT, acest tip de wavelet este scalat și translatat pentru a evidenția caracteristici la diferite rezoluții. O trăsătură definitorie a acestui wavelet este prezența unui parametru care permite ajustarea dimensiunii ferestrei Gaussiene în raport cu perioada semnalului sinusoidal [28].

## 2.2 Algoritmi de învățare profundă

Înainte de a aprofunda conceptul de învățare profundă, este importantă înțelegerea principiilor fundamentale ale învățării automate. Algoritmii de ML analizează seturi de date complexe și identifică tipare pe care le folosesc pentru a genera predicții sau recomandări, performanța acestora îmbunătățindu-se pe măsură ce lucrau cu mai multe date. Însă, metodele de învățare automată erau limitate în ceea ce privește capacitatea lor de a lucra cu date brute, mai exact pentru a dezvolta un sistem de recunoaștere a tiparelor era necesară intervenția umană pentru a extrage caracteristici și a transforma datele într-un format potrivit. Pentru a reduce acest aspect, a fost dezvoltat domeniul învățării profunde, capabil să învețe caracteristici direct din date, precum imagini, text sau audio. Deep learning este bazat pe conceptul de rețele neuronale artificiale (ANN), inspirate din modul în care funcționează creierului uman. Aceste rețele sunt formate din mai multe straturi de neuroni interconectați care procesează informația în etape succesive. Fiecare etapă determină sistemul să recunoască trăsături din ce în ce mai complexe și mai abstracte. De exemplu, un strat inițial poate detecta margini sau forme simple, în timp ce straturi mai profunde pot îmbina aceste forme pentru a identifica obiecte sau concepte. Asemenea ML, sistemele DL învață prin iterații, denumite epoci, ajustându-și parametrii pentru îmbunătățirea continuă a performanțelor [29], [30].

### 2.2.1 Rețele neuronale

Creierul uman, ca sursă de inspirație, a determinat dezvoltarea sistemelor artificiale de procesare a informației, precum neuronii artificiali. El este alcătuit din aproximativ 86-100 de miliarde de neuroni interconectați, aceștia fiind specializați în recepționarea, procesarea și transmiterea semnalelor electrice. Numărul de conexiuni sinaptice este de aproximativ 600 de ori mai mare, indicând importanța dependenței dintre neuroni pentru o funcționare corespunzătoare. O rețea neuronală artificială este o structură artificială bazată pe modul de funcționare a creierului uman, fiind alcătuite din multiple elemente de procesare sau neuroni artificiali organizați în straturi [31].

Una dintre cele mai simple forme a ANN este perceptronul, dezvoltat de Frank Rosenblatt în anii 1950-1960, care s-a inspirat din studiile lui Warren McCulloch și Walter Pitts. Aceștia au reușit în anul 1943, implementarea primului model neuronal artificial care funcționa ca un dispozitiv binar. Acesta stă la baza dezvoltării unor arhitecturi mult mai complexe, ce sunt alcătuite din mai multe straturi neuronale [31], [32].

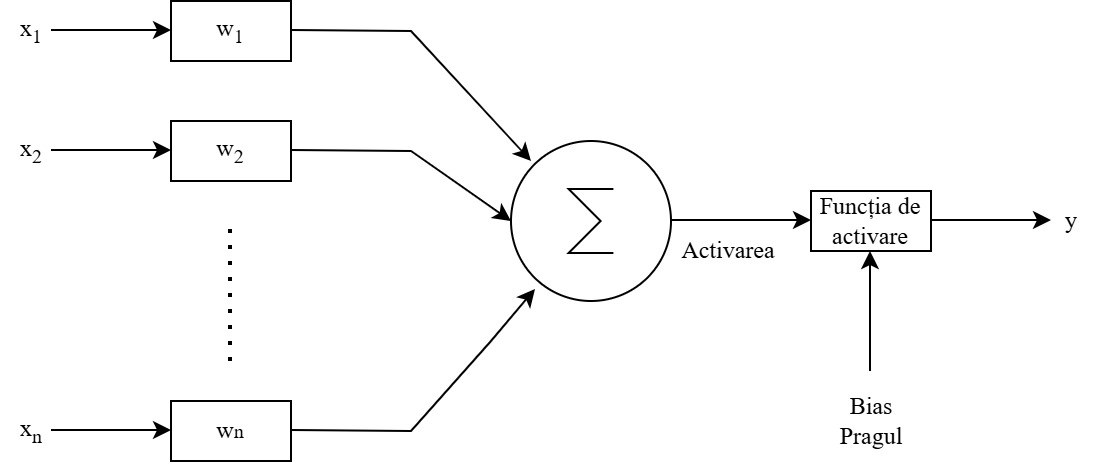


Figura 2. 2 Structura unui neuron artificial

Un neuron artificial funcționează prin preluarea mai multor intrări binare și produce o singură ieșire. Fiecăre intrare este asociată cu o pondere sinaptică, , care reflectă importanța datelor de intrare în decizia finală. Semnalul de intrare este procesat prin înmulțirea cu valorile ponderilor asociate, iar ulterior are loc o însumare globală a rezultatelor obținute. Sumei rezultate i se adaugă o constantă (bias) care contribuie la decizia de activare a neuronului. Ulterior, este aplicată o funcție de activare și este transmis mai departe un răspuns [31], [32].

Relația care caracterizează structura neuronului din figura 2.2 este:

(2.6)

Funcția *f* reprezintă funcția de activare, cea care decide dacă un neuron ar trebui „activat” sau nu, în funcție de datele de intrare. Mai exact, se referă la o operație matematică aplicată sumei rezultate care decide dacă datele sunt suficient de relevante pentru a influența rezultatul final [33].

Există mai multe tipuri de funcții matematice de activare care deservesc scopuri diferite în cadrul rețelelor neuronale. Funcția de activare binară (Binary step function) reprezintă o funcție ce acționează pe baza unui prag impus pentru activarea neuronilor și este potrivită unei probleme de clasificare cu două clase. Funcția de activare liniară nu aplică niciun fel de transformare asupra datelor, ceea ce limitează capacitatea de învățare a rețelei și face ca toate straturile să se rezume la operații liniare, anulând avantajele unei rețele profunde. Toate aceste limitări pot fi depășite prin utilizarea funcțiilor neliniare, care permit modelelor să învețe relații complexe și neliniare dintre datele de intrare și cele de ieșire [33].

Sigmoid reprezintă o funcție de activare care transformă datele de intrare în valori cuprinse în intervalul (0,1), ceea ce o face potrivită pentru modele care trebuie să exprime datele de ieșire sub forma unor probabilități. Tangenta hiperbolică este asemănătoare funcției sigmoid, diferența fiind că datele sunt convertite în valori cuprinse între (-1,1), un interval centrat în 0, astfel oferind o învățare mai eficientă. ReLU (Rectified Linear Unit) este o funcție de activare care returnează valoarea de intrare dacă acestea este pozitivă și 0 dacă aceasta este negativă, astfel activând doar un set de neuroni. Funcția Softmax este o extensie a funcției Sigmoid adaptată pentru clasificarea multi-clasă, convertind rezultatele în valori probabilistice care se însumează la 1. De obicei, Softmax este folosit în stratul final al unei rețele, deoarece permite identificarea clasei prin atribuirea unei probabilități maxime unei ieșiri, astfel oferind o predicție finală [33].

Alegerea unei funcții de activare într-o rețea poate influența semnificativ performanțele acesteia trebuie realizată ținând cont de tipul de predicție dorit și de arhitectura rețelei. În general, funcția ReLU este recomandată pentru utilizarea în cadrul straturilor intermediare, fiind adesea utilizată în rețelele neuronale convoluționale. Pe de altă parte, Tangenta hiperbolică și Sigmoid sunt mai des folosite în cadrul rețelelor neuronale recurente. În ceea ce priveșe stratul de ieșire, alegerea funcției depinde de natura sarcinii: pentru clasificare binară se utilizează funcția Sigmoid, regresia are nevoie de o funcție liniară și clasificarea multi-clasă necesită funcția Softmax [33].

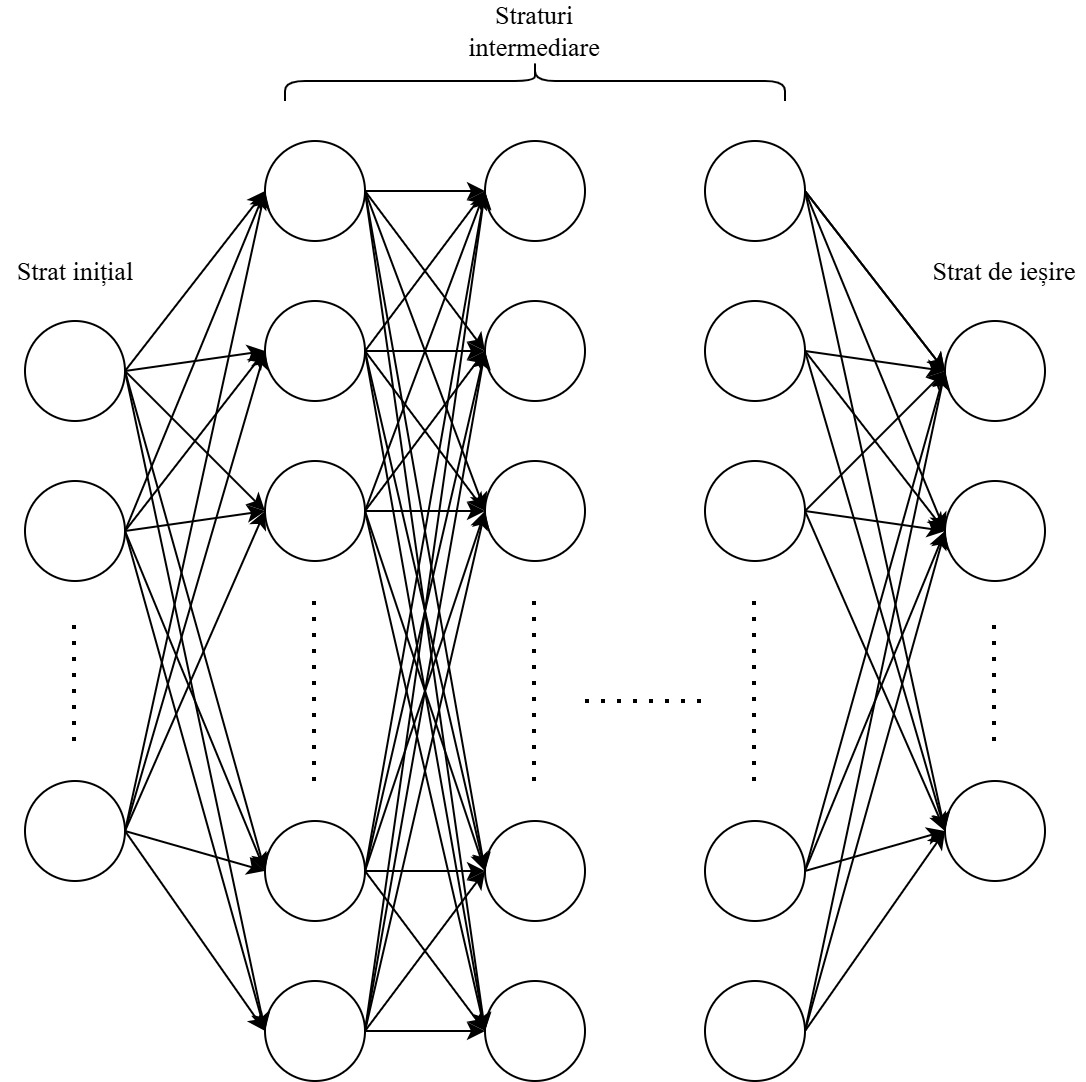


Figura 2.3 Arhitectura unei rețele neuronale

Rețelele neuronale artificiale sunt organizate, la un nivel de bază, în straturi, care reprezintă o componentă importantă în arhitectura unei rețele. Un strat este format din mai mulți neuroni care procesează datele în paralel. Cele mai utilizate rețele includ trei tipuri de straturi: stratul de intrare, stratul intermediar (ascuns) și cel de ieșire. Stratul de intrare constă în neuroni care preiau datele brute și inițiază procesul de învățare al rețelei. Straturile intermediare realizează majoritatea calculelor, facilitând astfel învățarea, iar stratul de ieșire generează rezultatele și oferă predicțiile finale ale rețelei [34].

Învățarea într-o rețea neuronală reprezintă procesul prin care rețeaua își ajustează automat valorile ponderilor pe baza unui set de date de antrenare, fără a fi necesară intervenția manuală asupra fiecărui parametru. Scopul principal este de a obține fenomenul de generalizare, adică abilitatea rețelei de a oferi predicții corecte pentru date care nu au fost întâlnite în timpul antrenării. În cadrul învățării supervizate, algoritmul compară rezultatul generat de rețea și rezultatul dorit, care este furnizat rețelei prin etichetarea corespunzătoare a datelor de intrare. Diferența dintre cele două valori reprezintă eroarea de adaptare care este utilizată ulterior pentru a corecta ponderile. Ajustările sunt realizate, folosind algoritmi de antrenare, care au scopul de a minimiza eroarea, astfel îmbunătățind precizia modelului [31].

Procesul fundamental care stă la baza învățării supervizate este algoritmul „backpropagation”. Acest algoritm aplică regula lanțului din calculul diferențial pentru a determina modul în care ajustarea parametrilor rețelei afectează eroarea totală, definită cu ajutorul unei funcții cost. La nivel matematic, obiectivul principal este calcularea gradientului funcției de cost în raport cu fiecare parametru [35].

Acest proces începe cu propagarea în direcția înainte, etapa care generează predicțiile pe baza datelor de antrenament, acestea trecând succesiv prin straturile care compun strucutura rețelei. Mai exact, propagarea în direcția înainte se referă la o succesiune de calcule matematice, în care ieșirea fiecărui strat devine intrarea pentru următorul strat. La începutul procesului de antrenare, valorile parametrilor, weights și bias, sunt inițializați aleator, în timp ce hiperparametrii precum: numărul de straturi, numărul de neuroni și funcțiile de activare sunt setați anterior și rămân neschimbați pe parcursul procesului. În timpul parcurgerii, un eșantion de intrare este introdus în stratul inițial, iar fiecare neuron transmite aceasta valoare, înmulțită cu valoarea unei ponderi, straturilor intermediare. Acestea calculează suma ponderată a datelor, aplică funcția de activare, iar rezultatul este transmis următorului strat. Acest flux se continuă de la stânga la dreapta, până la stratul de ieșire, unde predicția finală este realizată. Funcția cost transformă eroarea obținută într-o valoare numerică care reflectă tipul erorii. Există mai multe tipuri de funcții cost, fiecare potrivită pentru diverse sarcini. Funcția Eroare Medie Pătratică este frecvent utilizată pentru regresii, în timp ce unei sarcini de clasificare îi este potrivită funcția de pierdere „cross-entropy” [36].

Propagarea pe direcția înapoi se desfășoară în direcția opusă propagării înainte, începând cu ultimul strat al rețelei. În această etapă, funcția cost este derivată pentru a observa cum fiecare parametru contribuie la eroarea totală. Scopul final al clasificării este ca rețeaua să fie ajustată astfel încât neuronul care corespunde clasei corecte să genereze o valoare apropiata de 1, iar celelalți neuroni valori apropiate de 0. Procesul include calcularea derivatelor parțiale atât în raport cu ponderile, cât și în raport cu valorile de activare ale neuronilor. Rețeaua calculează un set de gradienți, care reprezintă modul în care variații ale parametrilor afectează eroarea. [36]

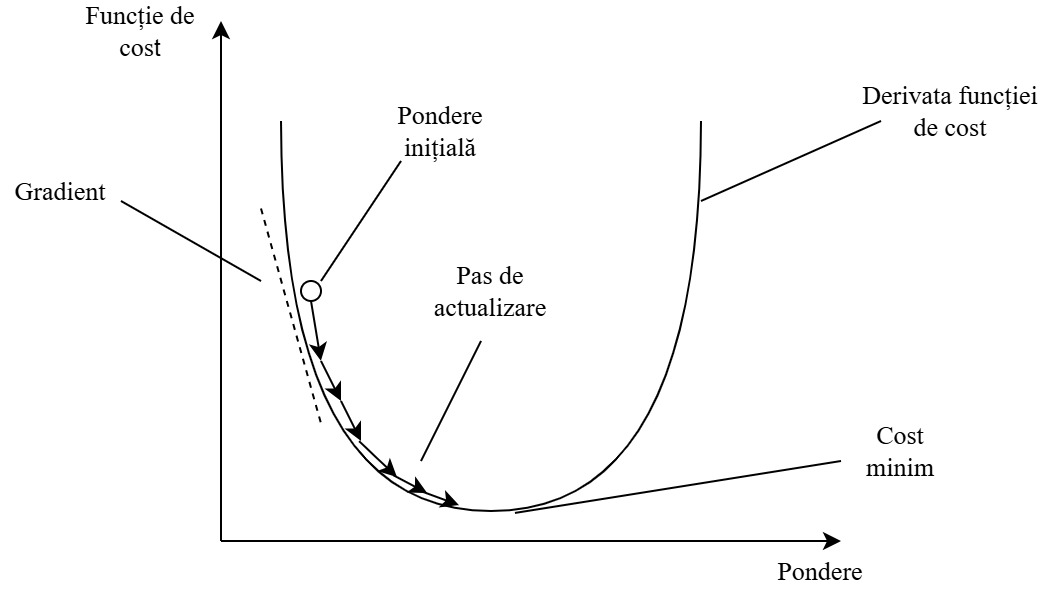


Figura 2.4 Algoritmul gradientului descendent

Odată ce gradientul funcției cost a fost obținut, modelul este optimizat folosind metoda gradientului descendent. Acest algoritm funcționează prin ajustarea parametrilor în direcția opusă gradientului, ceea ce duce la o reducere în cadrul pierderilor. Gradientul calculat conține derivatele parțiale, permițând determinarea direcției corecte în care fiecare parametru trebuie modificat. Fiecare actualizare reprezintă un pas de învățare în cadrul procesului de antrenare. Scopul acestui proces iterativ este de a atinge o valoarea minimă a funcției cost [37].

Rata de învățare este un hiperparametru care determină mărimea fiecărui pas de actulizare în timpul algoritmului gradientului descendent. Deoarece rețelele neuronale se bazează pe funcții de activare neliniare, gradientul rezultat poate avea forme complexe, în care un pas inițial în direcția corespunzătoare poate deveni ineficient dacă este prea mare. O rată de învățare mică permite actualizări precise și reduce riscul de a depăși soluția optimă, dar necesită un număr mai mare de iterații, astfel procesul de învățare fiind unul mai lent. O rată de învățare mai mare accelerează învățarea, dar există un risc ridicat de depășire a minimului funcției de pierdere, astfel împiedicând convergența modelului [36].

Un aspect important al algoritmului gradientului descendent este frecvența cu care sunt actualizați parametrii modelului. Folosind metoda Batch Gradient Descent, gradienții sunt calculați pentru întreg setul de date de antrenament. Acest lucru poate fi ineficient în cazul seturilor de date foarte mari deoarece implică un volum ridicat de calcule. Metoda Stochastic Gradient Descent (SGD) presupune realizarea actualizării pe baza unui singur eșantion. Deși mai rapidă, această metodă poate introduce mai multe fluctuații în cadrul funcții de pierdere. Mini-Batch Gradient este o alternativă care împarte setul de date în grupuri de dimensiune fixă. Această metodă reduce atât cerințele de memorie și timpul de procesare, cât și instabilitatea prezentă în SGD [37].

În timpul procesului de antrenare, setul de date este împărțit în 3 subseturi: antrenare, validare și testare. Setul de antrenare cuprinde cel mai mare procent din totatul datelor și este utilizat pentru ajustarea ponderilor, astfel încât rețeaua să învețe tiparele din date. Setul de validare are rolul de a observa capacitatea de generalizare a modelului și contribuie la selecția arhitecturii optime sau hiperparametrilor necesari pentru obținerea celor mai bune performanțe. Setul de test este folosit după finalizarea antrenării, pentru evaluarea performanțelor reale ale rețelei. În cadrul modelului poate apărea fenomenul de „underfitting”, atunci când rețeaua își pierde capacitatea de a învăța corespunzător. De asemenea, există și fenomenul de „overfitting”, atunci când modelul învăță bine din datele antrenament, dar nu are performanțe bune pe datele de validare. Aceste probleme pot fi controlate prin ajustarea corespunzătoare a hiperparametrilor sau utilizarea tehnicilor de regularizare [31], [38].

### 2.2.2 Rețele neuronale convoluționale

Similar rețelelor neuronale artificiale, rețelele neuronale convoluționale (CNN) sunt alcătuite din neuroni care învață și își îmbunătățesc performanțele pe măsură ce intră în contact cu date noi. Însă, rețelele convoluționale sunt diferite de rețelele neuronale artificiale prin faptul că, arhitectura acestora conține filtre și operații de grupare, utile pentru învățarea caracteristicilor vizuale. De aceea, CNN sunt potrivite sarcinilor în care sunt clasificate imagini, detectate obiecte sau pentru recunoaștere facială. Totuși, ele pot utilizate pentru a prelucra și alte tipuri de date, cum ar fi semnalele audio sau vocale [39].

CNN sunt structurate pe baza a trei tipuri de straturi principale: de convoluție, de grupare și complet conectate. Stratul convoluțional este primul strat și este responsabil de extragerea trăsăturilor simple. Pe măsură ce datele sunt transmise prin alte straturi convoluționale și straturi de grupare, rețeaua reușește identificarea unor caracteristici mai complexe. Stratul final este cel complet conectat, care utilizează trăsăturile extrase pentru a realiza clasificarea finală a datelor procesate [40].

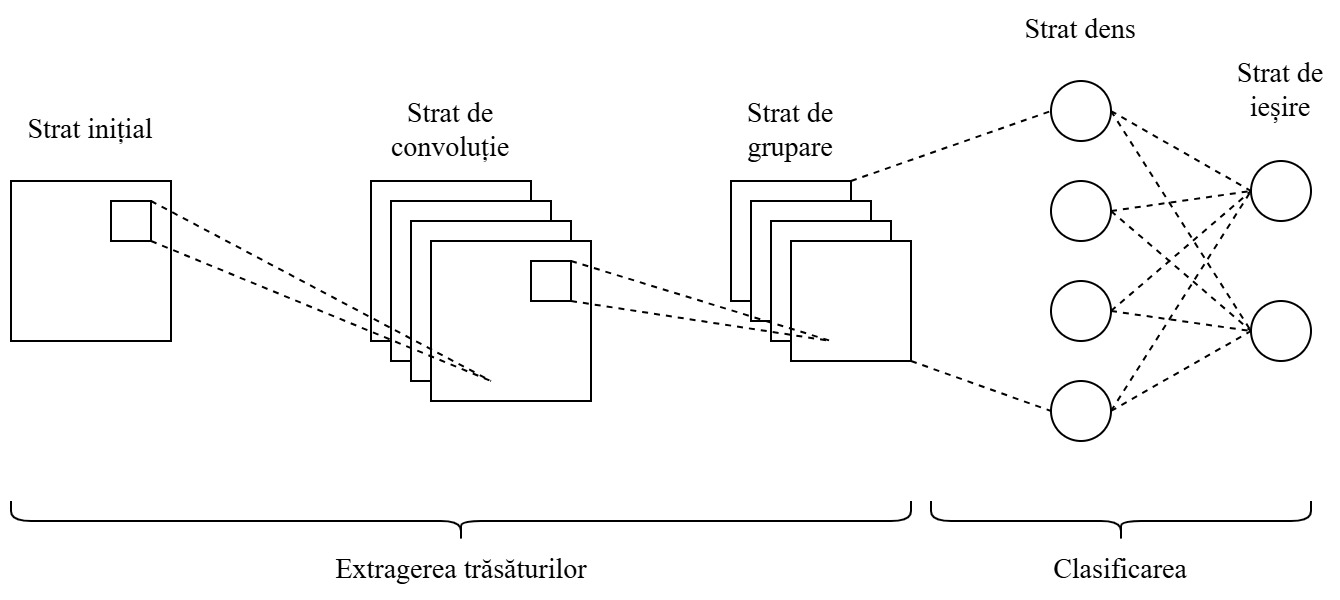


Figura 2.5 Arhitectura de bază a unei rețele neuronale convoluționale

Stratul de convoluție reprezintă componenta esențială în cadrul CNN. Acesta procesează datele de intrare cu ajutorul filtrelor/nucleelor care se deplasează peste imagine pentru a detecta trăsături. Fiecare filtru, reprezentat de o matrice alcătuită din ponderi, realizează produsul scalar cu valorile corespunzătoare datelor de intrare pentru a genera o „hartă” a caracteristicilor. Acest proces este influențat de hiperparametri: numărul de filtre (care determină profunzimea rezultatului), pasul cu care se deplasează filtrul și „padding” (pentru a gestiona marginile în cazul unei imagini) [40].

Straturile de grupare reduc dimensiunea rezultatului obținut în urma convoluției și minimizează numărul de parametri, utilizând filtre, care spre deosebire de straturile covoluționale, nu conțin valori ale ponderilor. Straturile de grupare se împart în două tipuri principale: cele de tip „max pooling”, acestea fiind cele mai utilizate și selectează valoarea cea mai mare din fiecare zonă și „average pooling”, care calculează o valoare medie. Straturile de grupare sunt utilizate deoarece oferă o generalizare mai bună și reduc riscul de apariție a supraînvățării, dar din cauza faptului că este un proces distructiv, pot duce la pierderea excesivă a datelor. Din acest motiv, mărimea filtrului din cadrul straturilor de grupare, care controlează gradul de micșorare a datelor, reprezintă un parametru important [39], [40].

Straturile dense, cunoscute și ca straturi complet conectate, sunt straturi în care fiecare neuron este conectat la toți neuronii din stratul anterior și cel următor. Fiecare neuron aplică o transformare liniară datelor de intrare, urmată de o funcție de activare. Acest tip de conexiune este similar modului în care neuronii sunt aranjați în cadrul rețelelor neuronale artificiale clasice [39], [41].

### 2.2.3 CoAtNet

CoAtNet reprezintă un model hibrid care combină rețelele neuronale convoluționale cu arhitectura de tip Transformer. [42]

Transformer reprezintă un tip de arhitectură caracteristică rețelelor neuronale, utilizată pentru sarcini din domeniul procesării limbajului și funcționează pe baza mecanismului de „self-attention. Mecanismul permite învățarea dependențelor dintre elementele unei secvențe, oferind posibilitatea ca aceastea să interacționeze între ele. Arhitectura are structura „encoder-decoder”, în care „encoderul” are rolul de a genera reprezentări care să ilustreze contextul pentru datele inițiale, în timp ce „decoderul” realizează secvența de ieșire. Urmând această metodă, arhitectura Transformer poate oferi o perspectivă globală asupra datelor, ceea ce facilitează prelucrarea datelor vizuale [43].

Algoritmul CoAtNet îmbină vertical straturi convoluționale și straturi de atenție, pentru a îmbunătăți capacitatea și eficiența modelului. Arhitectura prezintă blocuri de tip MBConv, care utilizează convoluția „depthwise”, și are o structură organizată în 5 etape, în care rezoluția spațială este redusă treptat. Prima etapă constă într-o operație de convoluție formată din 2 straturi, ulterior urmând etapa în care sunt aplicate blocurile MBConv, necesare din cauza dimensiunii mari a datelor, care ar face ineficientă aplicarea mecanismului de atenției. Începând cu următoarea etapă, se pot implementa atât blocuri convoluționale, cât și cele de tip Transformer, în funcție de cerințele arhitecturii. Această organizare se datorează faptului că straturile de convoluție inițiale sunt potrivite pentru detectarea trăsăturilor de bază, în timp ce în etapele superioare, straturile de atenție se pot concentra asupra dependențelor globale [42].

Blocul MBConv a fost prima oară introdus în arhitectura MobileNetV2 și este un modul din cadrul rețelelor neuronale creat pentru a reduce calculele de procesare fără a pierde din performanță. Structura sa include un strat de expansiune care mărește dimensiunea datelor, urmat de o convoluție „depthwise”. Spre deosebire de metodele clasice de convoluție care îmbină informațiile din cadrul tuturor canalelor, convoluția „depthwise” procesează datele prin aplicarea unui filtru pentru fiecare canal individual. Ulterior, este utilizat un strat de proiecție pentru a readuce datele la dimensiunea inițială. Dacă dimensiunile datelor de intrare și celor de ieșire corespund, se aplică o conexiune reziduală pentru a facilita o antrenare mai stabilă și mai eficientă. [44]

Totuși, algoritmul CoAtNet prezintă și câteva limitări. Aplicarea directă a mecanismului de atenție pe imagini cu o rezoluție înaltă poate fi costisitor din punct de vedere al calculelor de procesare, motiv pentru care este necesară reducerea dimensiunii datelor. Acest lucru însă, poate cauza pierderi considerabile de informații, în special la nivelul trăsăturilor de bază. Arhitectura de tip Transformer necesită cantități mari de date pentru o antrenare corespunzătoare, iar în contextul antrenării cu un set de date limitat, există riscul apariției fenomenului de supraînvățare, din cauza lipsei bias-urilor specifice rețelelor convoluționale [42].

### 2.2.4 Procesarea limbajului natural

Procesarea limbajului natural (NLP) se referă la un domeniu în care sunt îmbinate concepte de lingvistică cu metode statistice sau bazate pe tehnici de învățare automată și care oferă computerelor abilitatea de a înțelege și de a genera limbaj uman scris sau vorbit. Există trei metode principale prin care NLP este abordat, și anume: poate fi bazat pe reguli predefinite, NLP statistic, care utilizează algoritmi de învățare automată și NLP bazat pe rețele neuronale [45], [46].

Procesarea limbajului natural funcționează urmând o anumită structură care începe prin preprocesarea textului, o etapă care include „tokenization”, mai exact separarea în cuvinte sau propoziții, ștergerea cuvintelor uzuale și eliminarea caracterelor cum ar fi punctuația sau numerele. Urmează etapa de extragere a caracteristicilor, în care textul este convertit în numere pentru a putea fi interpretat. Există mai multe tipuri de tehnici care permit diferite aspecte: înregistrarea importanței sau apariției cuvintelor și observarea relațiilor semantice dintre cuvinte. Ulterior, este realizată analiza textului care presupune interpretarea datelor în vederea obținerii unor informații mai relevante. Ultima etapă se referă la antrenarea modelelor de învățare automată pentru a procesa datele din care să invețe tipare și să genereze predicții [46].

NLP are diverse utilități și este integrat în mai multe aplicații, printre care se numără: sistemele Siri sau Alexa, extragerea opiniilor din recenzii sau rețelele de socializare, traducerea textelor, instrumente pentru scrierea și editarea conținutului etc. [45].

### 2.2.5 Metrici de evaluare a modelului

Pentru a putea analiza performanțele modelele de învățare automată, mai exact capacitatea acestora de a realiza predicții și de a generaliza, metricile de evaluare sunt necesare procesului. Acestea sunt alese în funcție de cerințele problemei, însă pentru clasificarea datelor, cele mai utilizat metrici sunt acuratețea, precizia, senzitivitatea și scorul f1. Evaluarea trebuie să fie realizată pe un set de date de test, pe care modelul nu le-a mai văzut, pentru obținerea unor rezultate cât mai realiste. Metricile de evaluare contribuie semnificativ la optimizarea și ajustarea modelului în cadrul procesului de dezvoltare.

Calcularea metricilor necesită definirea următorilor termeni:

TP (True Positive) = predicție pozitivă, valoare reală pozitivă

TN (True Negative) = predicție negativă, valoare reală negativă

FN (False Negative) = predicție negativă, valoare reală pozitivă

FP (False Positive) = predicție pozitivă, valoare reală negativă

Acuratețea reprezintă procentul predicțiilor corecte și este definită matematic conform relației: [31]

(2.7)

Un model ideal ar obține o acuratețe de 100%, adică ar realiza exclusiv predicții corecte. Acuratețea este o metrică ce ia în considerare toate elementele din matricea confuziilor și este frecvent utilizată atunci când setul de date este echilibrat, adică fiecare clasă are un număr similar de exemple. Pentru un set de date dezechilibrat, atunci când un rezultat FP sau FN are un impact mai mare decât celălalt, alte tipuri de metrici sunt recomandate.

Matricea confuziilor reprezintă o metodă de vizualizare, mai exact un tabel, care evidențiază în ce măsură predicțiile acestuia coincid cu valorile reale. Aceasta arată, pentru fiecare clasă, câte exemple au fost clasificate corect și cât au fost etichetate greșit [31].

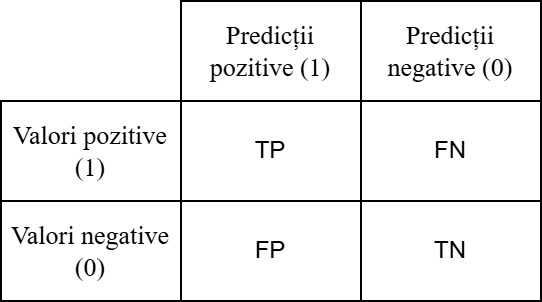


Figura 2.6 Matricea confuziilor

În cadrul unei matrici de confuzie, fiecare linie indică clasele reale, în timp ce coloanele reprezintă clasele prezise de model. Cea mai simplă versiune a acesteia există în cadrul clasificării binare, structura sa fiind prezentată în figura 2.6. Pentru clasificările multi-clasă, matricea este mai extinsă și are următoarea configurație: diagonala principală prezintă valorile TP, adică predicțiile corecte pentru fiecare clasă, în timp ce restul restul valorile pot fi considerate TN, FP sau FN. În scenariul în care modelul clasifică corect toate exemplele, toate valorile se vor regăsi pe diagonală, iar celelalte vor avea valoarea 0 [47].

Precizia reprezintă proporția predicțiilor pozitive care corespund valorilor reale pozitive. Este calculată cu ajutorul formulei:

(2.8)

Precizia reprezintă o metrică important în situațiile în care rezultatele FP pot avea un impact major. O valoarea ridicată a preciziei indică faptul că predicțiile sunt corecte, minimizând numărul de clasificări eronate [48].

Senzitivitatea sau rapelul măsoară capacitatea unui model de a detecta corect toate valorile pozitive dintr-un set de date. O valoare ridicată a senzitivității poate duce uneori la o scădere a preciziei, crescând numărul de valori FP. Astfel, cele două metrici sunt adesea utilizate împreună pentru o evaluare echilibrată. Senzitivitatea este calculate conform formulei: [48]

(2.9)

Între precizie și senzitivitate există un compromis, îmbunătățirea uneia poate duce la degradarea celeilalte. Pentru a gestiona acest dezechilibru, este utilizat scorul F1, care reprezintă media armonică dintre cele două metrici. Scorul F1 acordă o importanță egală, atât senzitivității, cât și preciziei, ceea ce nu este întotdeauna ideal. De aceea, există alte versiuni, cum ar fi scorul F0 sau F2, care permit ajustarea evaluării în funcție de cerințele aplicației. Scorul F1 este definit astfel: [48]

(2.10)

## 2.4 Tehnologii software utilizate

### 2.4.1 Python

Toate codurile din cadrul proiectului au fost create folosind Python, care este un limbaj de programare open-source[[3]](#footnote-3), ce permite două tipuri diferite de programare, și anume procedurală și orientată pe obiecte. Python poate fi caracterizat de mai multe aspecte, însă el este cunoscut pentru simplitate, sintaxă clară și o bibliotecă extinsă ce conține numeroase funcții și module [49].

Având în vedere că este un limbaj de nivel înalt, aceasta generalizează detaliile legate de hardware și simplifică procesul de scriere. O altă caracteristică a limbajului Python este că acesta include atât mai multe structuri de date predefinite, cum ar fi liste, dicționare sau seturi, cât și posibilitatea de descriere dinamică ceea ce permite determinarea automată a variabilelor [49].

Python este adesea utilizat în dezvoltarea de aplicații web și software, pentru procesarea imaginilor și conținutului audio, dar în special, în domeniul inteligenței artificiale, pentru care există numeroase biblioteci specializate [49].

### 2.4.2 Pytorch

PyTorch reprezintă un framework[[4]](#footnote-4) de învățare profundă care îmbină flexibilitatea programării dinamice cu performanța necesară algoritmilor de învățare automată. Spre deosebire de framework-urile tradiționale, precum TensorFlow sau Caffe, PyTorch permite utilizatorilor să interacționeze direct cu Python [50].

Un avantaj al PyTorch este că structura modelului poate fi modificată dinamic, în timpul execuției, ceea ce facilitează ajustarea arhitecturii modelului. În plus, una dintre funcționalitățile incluse în PyTorch este calculul automat al derivatelor, esențial în procesul de antrenare [51].

PyTorch obține performanțe ridicate în procesarea calculelor datorită componentei de bază implementată în limbajul C++ și unui sistem de gestionare a memoriei optimizat pentru folosirea mai multor GPU [50].

Peste 70% din proiectele de cercetare în domeniul AI folosesc PyTorch. [52] Acesta oferă suport pentru numeroase arhitecturi de rețele neuronale, începând cu modele de bază cum ar fi regresia liniară până la rețele convoluționale și modele de tip Transformer.

### 2.4.3 Librosa

Librosa este o bibliotecă Python specializată în analiza semnalelor audio și muzicale. Are ca scop principal, procesarea semnalului audio pentru o analiză eficientă, utilizând o interfață Python. La bază, Librosa permite utilizatorilor să încarce, să manipuleze și să analizeze date audio, prin funcționalități cum ar fi extragerea caracteristicilor, vizualizarea semnalelor sau aplicarea altor transformări audio. Este o bibliotecă construită exclusiv în Python și se integrează eficient cu mediul acestuia, colaborând cu biblioteci precum NumPy sau matplotlib. Unul dintre avantajele prezente în Librosa este că permite personalizarea funcțiilor implicite, un aspect util pentru experimente și cercetare [53].

Biblioteca Librosa este utilizată în aplicații precum clasificarea genurilor muzicale, sisteme de recomandări muzicale sau preprocesarea audio pentru modele de învățare profundă [53].

### 2.4.4 Streamlit

Streamlit reprezintă o bibliotecă Python open-source, destinată construirii aplicațiilor web interactive. Biblioteca Streamlit este caracterizată de simplitate și flexibilitate, prin faptul că poate fi utilizată pentru diverse aplicații, precum vizualizarea datelor sau modele de învățare profundă. De asemenea, oferă și o serie de instrumente cu scopul de a crea grafice, de a le afișa, precum și de a permite utilizatorilor să exploreze date în timp real. [54]

# *Capitolul 3. Implementarea practică*

## 3.1 Structura proiectului

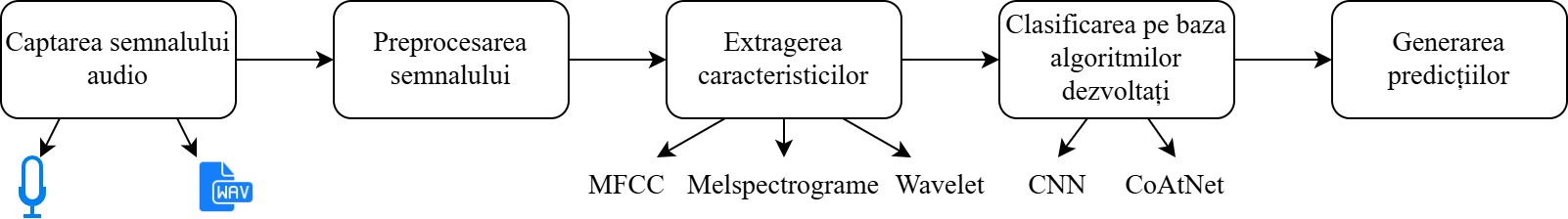


Figura 3.1 Etapele necesare realizării proiectului

Implementarea practică a proiectului include mai multe etape, astfel:

Procesul începe cu etapa de captare a semnalului, mai exact etapa de creare a bazei de date prin realizarea înregistrărilor cu ajutorul unui microfon situat la o distanță fixă față de tastatură, care apoi sunt stocate și transmise către etapa de preprocesare.

Preprocesarea semnalelor are ca scop îmbunătățirea calității și gradului de consistență a înregistrărilor pentru a putea fi utilizate în etapa de extragere a caracteristicilor. Această etapă presupune, în principiu, eliminarea zgomotului pe cât mai mult posibil prin aplicarea unor filtre sau algoritmi, normalizarea amplitudinilor și segmentarea înregistrărilor. Este necesară normalizarea semnalelor, deoarece astfel există mai puține variații prin ajustarea amplitudinilor formei de undă.

Extragerea caracteristicilor reprezintă o etapă fundamentală în cadrul proiectului, deoarece ea oferă datele necesare pentru algoritmi în vederea clasificării semnalelor. Etapa prespune generarea unor reprezentări vizuale ale semnalelor precum melspectrograme sau bazate pe coeficienții Mel-Cepstrali și, respectiv utilizarea Transformatei Wavelet.

Următoarea etapă este reprezentată de antrenarea algoritmilor cu ajutorul imaginilor obținute în urma pasului anterior. Fiecare imagine corespunde unei taste și este etichetată corespunzător. Prin realizarea acestei etape, este dorită obținerea unor modele capabile de a generaliza astfel încât să poată clasa și semnale noi.

În urma clasificării datelor, sunt obținute predicții pentru fiecare semnal acustic, formând ulterior secvențe de caractere ce reprezintă datele tastate inițial. Cum în cadrul acestora pot exista caractere greșite, este aplicată o tehnică bazată pe NLP, care sugerează cuvinte alternative pe baza unor probabilităților oferite de model. Cu scopul de a prezenta aceste rezultate, a fost realizată o interfață web, care funcționează cu ajutorul algoritmilor implementați în etapa de generare a predicțiilor.

Proiectul este structurat în 5 fișiere, fiecare reprezentând una dintre etapele din cadrul procesului descris anterior, astfel: features.py, care include procesarea semnalelor (Anexa 1), cnn.py (Anexa 2), coatnet.py (Anexa 3), care includ arhitecturile și funcțiile de antrenare, validare și testare, predict.py (Anexa 4), app.py (Anexa 5) și plot.py și matrix.py (Anexa 6).

## 3.2 Descrierea setului de date

Baza de date este alcătuită din aproximativ 56 de înregistrări pentru fiecare tasta, însă pot exista diferențe între clase din cauza procesului de înregistrare manuală. Înregistrările au fost realizate într-un mediu fără zgomot, folosind un smartphone, mai exact modelul iPhone 11, amplasat la o distanță fixă față de tastatură și pe un material de microfibră pentru a reduce vibrațiile. Acestea conțin mai multe apăsări ale tastelor, din acest motiv fiind necesar un algoritm de separare care funcționează prin detectarea creșterilor energetice din cadrul formei de undă. Rata de eșantionare a microfonului utilizat este de 48000 Hz.

Tastatura folosită pentru colectarea datelor este modelul Dark Alien K870 RGB, aceasta fiind o tastatură mecanică care are tastele dispuse conform standardului QWERTY. O caracteristică definitorie a tastaturii este prezența comutatoarelor albastre, care oferă un sunet specific. A fost utilizat acest model de tastatură, deoarece față de alte tipuri de tastaturi care pot fi mai silențioase, aceasta permite înregistrarea unor sunete mai clare și mai ușor de recunoscut.

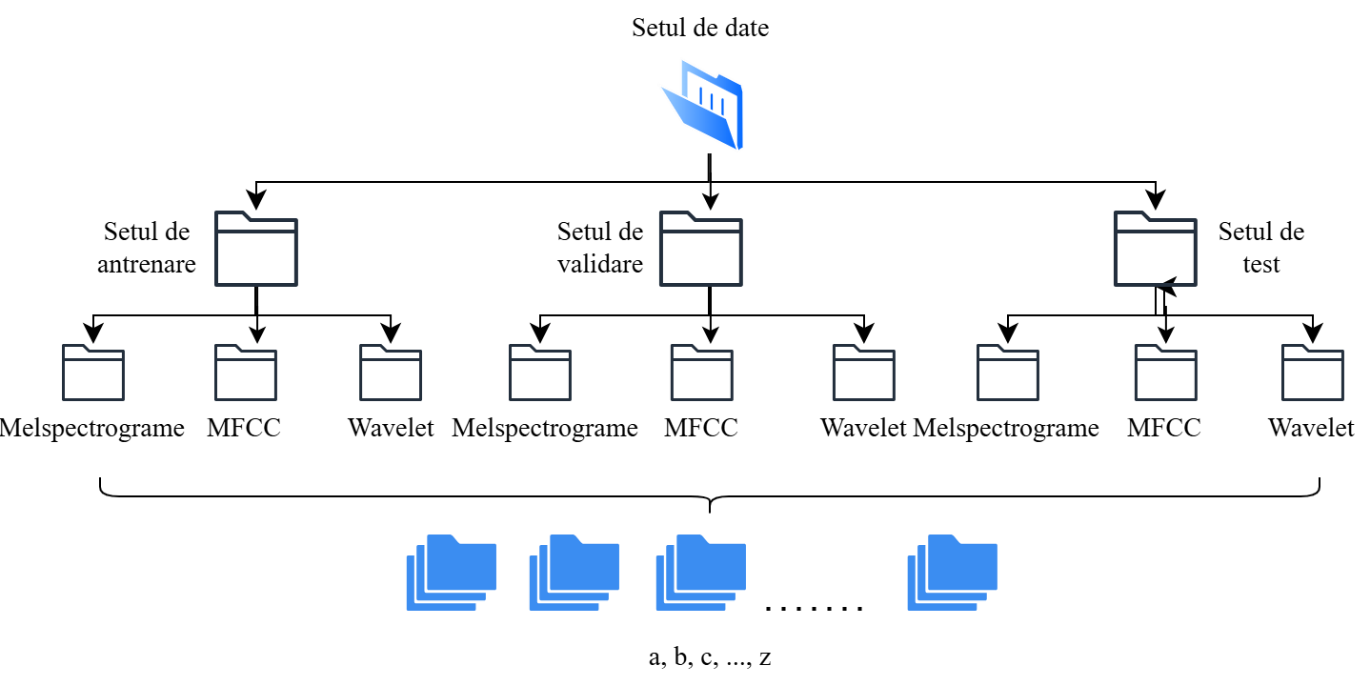


Figura 3.2 Structura setului de date

Inițial înregistrările obținute sunt în format .m4a, datele fiind comprimate, fapt ce poate duce la pierderea unor detalii ale sunetului. Pentru a putea fi utilizate în cadrul algoritmilor de învățare profundă, este necesară convertirea acestora în format .wav, după care sunt stocate și dispuse în directoare organizate în funcție de clase. Clasele se referă la caracterele incluse în baza de date, iar în cadrul proiectului, acestea sunt limitate doar la literele incluse în alfabetul englez.

Ulterior, fișierele audio segmentate sunt utilizate pentru a realiza imaginile ce vor alcătui seturile de date folosite în procesul de învățare. Sunt generate, în mod egal pentru fiecare clasă, în jur de 1448 de imagini pentru fiecare tip de reprezentare vizuală, astfel asigurând un număr consistent de exemple pentru a menține setul de date echilibrat. Similar înregistrărilor segmentate, imaginile sunt stocate sub aceeași formă, în directoare etichetate cu numele claselor corespunzătoare.

Structura setului de date final utilizată în procesul de antrenare este prezentată în figura 3.2. Se observă că, inițial setul este împărțit în subseturi folosite în cadrul etapelor incluse în procesul de învățare, iar apoi sunt separate în directoarele specifice tipului de reprezentare și respectiv clasei.

## 3.3 Analiza semnalelor audio

Analiza semnalelor reprezintă un pas important înainte de a extrage caracteristicile, iar pentru a avea o imagine generală asupra caracteristicilor și a diferențelor dintre sunetele tastelor, este realizată o analiză în domeniul timp, dar și în domeniul frecvență. Semnalele emise de taste pot varia considerabil, fiind influențate de mai mulți factori: stilul dactilografierii, echipamentele utilizate etc.

Procesul începe cu importarea înregistrărilor, utilizând funcția *load* din cadrul bibliotecii *librosa*, fără a schimba rata de eșantionare standard a microfonului utilizat. Este necesar ca rata de eșantionare să fie de cel puțin două ori mai mare decât frecvența maximă a semnalului, conform teoremei lui Nyquist, deci rata de eșantionare de 48000 Hz poate ilustra frecvențe de până la 24000 Hz.

După ce semnalele sunt importate, următorul pas se referă la normalizarea acestora, mai exact împărțirea fiecărui eșantion al semnalului la valoarea maximă absolută a amplitudinii. Astfel, valorile sunt păstrate în intervalul [-1,1]. Urmează aplicarea funcției *reduce\_noise* din modulul *noisereduce*, care elimină zgomotul din cadrul înregistrărilor, oferind o versiune mai calitativă a semnalului. Datorită faptului că înregistrările conțin mai multe apăsări de taste, este necesar implementarea unui algoritm pentru a realiza segmentarea acestora.

Scopul algoritmului implementat este de a obține separat fiecare apăsare a tastei pentru a fi salvate în fișiere .wav de o lungime fixă. Este utilizată funcția *onset.detect* pentru a identifica momentele de timp în care există o creștere bruscă în cadrul energiei. Acest lucru poate fi subiectiv, deoarece procesul de identificare a momentelor este influențat de parametrii funcției. Parametrul *delta* este considerat un prag, o valoare mai scăzută poate duce la detectarea unor valori fals pozitive, iar o valoare mai mare face procesul de detecție unul mai strict. Parametrii *pre\_max* și *post\_max* se referă la fereastra în care poate fi detectată o apăsare a tastei. În cadrul lucrării, parametrii au fost setați astfel: *delta* = 0.2, *pre\_max* = 10 și *post\_max* = 10. Pentru a nu permite detecția aceleași apăsări de mai multe ori, a fost adăugat un pas care să impună o distanță de 0,3 s între două apăsări succesive, iar durata înregistrărilor este de 0,3 s, pentru a asigura uniformitatea setului de date. Ultimul pas este reprezentat de salvarea înregistrărilor segmentate în format .wav pe care le stochează în directoare denumite în funcție de clasele existente.

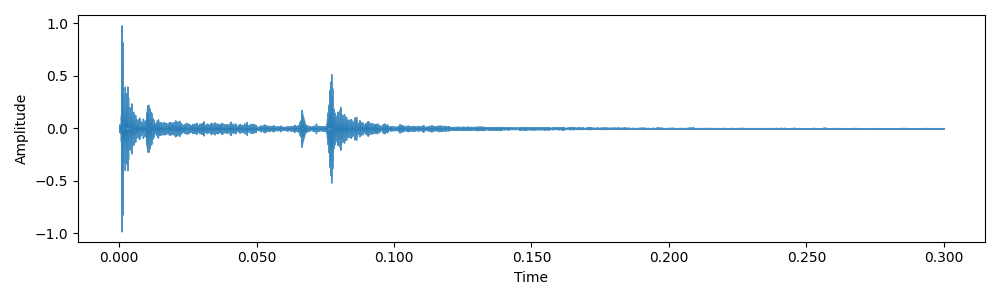


Figura 3.3 Forma de undă pentru tasta „q”

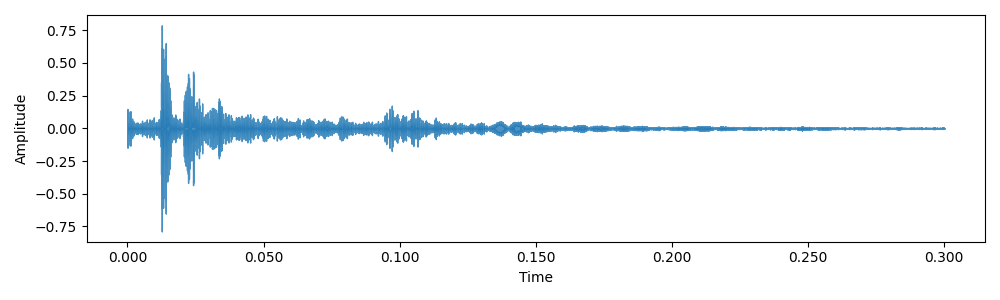


Figura 3.4 Forma de undă pentru tasta „m”

Cum principalul factor care influențează sunetele tastelor este poziția acestora pe placa tastaturii, vom analiza formele de undă pentru tastele *m* și *q*. Tasta *q* este situată în colțul din stânga sus, iar tasta *m* este poziționată în colțul din dreapta jos.

Din figura 3.3, se poate observa că momentul apăsării tastei *q* este caracterizat de o amplitudine mare, apropiată de valoarea maximă normalizată, ceea ce indică un sunet mai puternic față de sunetul apăsării tastei *m*. De asemenea, figura 3.4 ilustrează o distribuție energetică care scade treptat, fapt ce indică prezența oscilațiilor mecanice. În schimb, distribuția energiei pentru tasta *q* se concentrează în primele 80 ms și scade brusc ulterior. O ultimă diferență este reprezentată de faptul că, în cadrul primei figuri, există din nou a creștere bruscă în amplitudine, mai exact în momentul eliberării tastei, lucru care este mai puțin prezent în cazul tastei *m*.

Reprezentările în timp permit vizualizarea structurii generală a semnalelor, cum ar fi durata și amplitudinea. Însă, nu oferă cea mai bună bază pentru a crea un set de date optim pentru procesul de învățare, deoarece este limitat din punct de vedere al caracteristicilor vizuale care să diferențieze tastele între ele. În schimb, reprezentările frecvențiale pot furniza un set de date mai relevant pentru acest tip de semnale acustice.

Pentru a trece în domeniul spectral, am aplicat funcția *fft* din biblioteca *numpy* care creează un vector de numere complexe ce reprezintă componentele de frecvență ale semalului. Deoarece este dorită o reprezentare bidimensională a spectrului, este importantă utilizarea funcției *abs* pentru valorile complexe obținute, deoarece aceasta calculează valorile reale ale amplitudinilor fiecărei frecvențe.

Figura 3.5 prezintă o componentă spectrală proeminentă în jurul frecvenței de 7000 Hz, lucru care sugerează că tasta *q* produce un sunet cu frecvența relativ ridicată. Acest fapt se poate datora fie aplicării unei forțe mai mari în momentul apăsării, fie poziționării tastei. În figură se poate observa faptul că și restul componentelor sunt destul de pronunțate și nu depășesc frecvența de 10000 Hz. Tasta *m*, în schimb, are un spectru mai larg de frecvențe, cu o componentă dominantă în cadrul frecvențelor joase. Corespondent reprezentării în domeniul timp, distribuția energetică a frecvențelor indică un sunet mai slab și rezonant.

Comparațiile anterioare demonstrează diferențele ce pot exista între sunetele tastelor și influența factorilor asupra acestora. Tasta *q* a generat un sunet mai intens, cu energie concentrată în zona frecvențelor mai înalte, în timp ce tasta *m* a produs un sunet caracterizat frecvențe mai joase, fiind mai resonant.

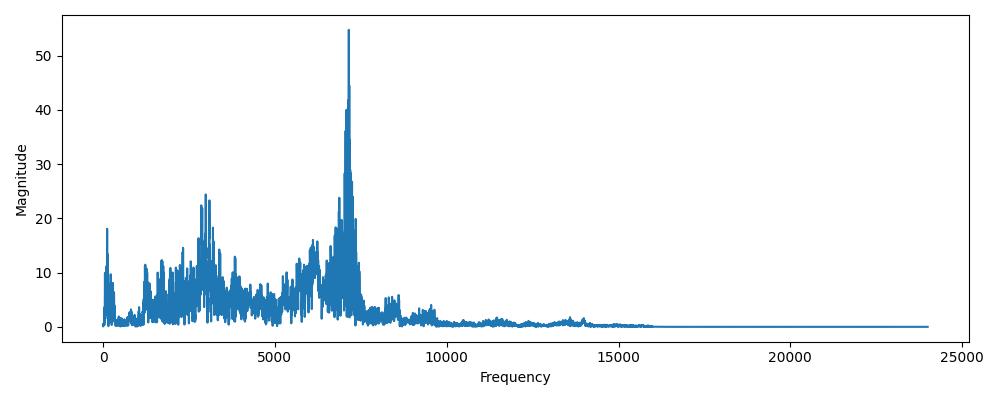


Figura 3.5 Analiza în domeniul frecvență pentru tasta „q”

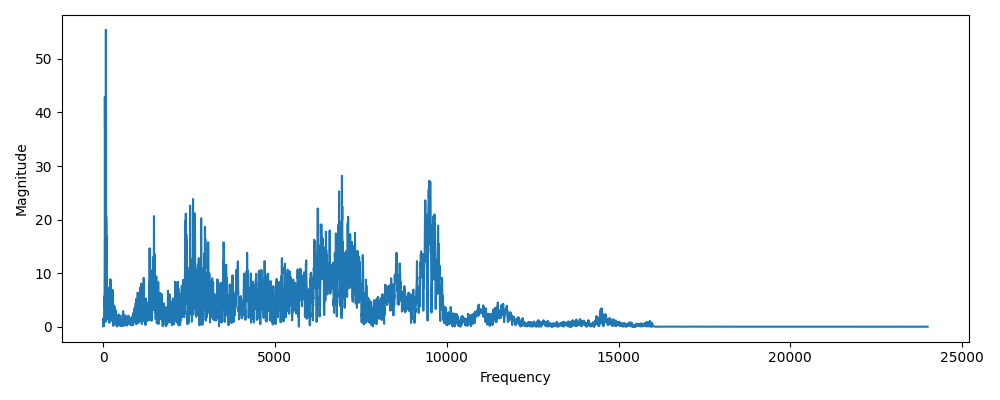


Figura 3.6 Analiza în domeniul frecvență pentru tasta „m”

## 3.4 Extragerea caracteristicilor

Modelele de învățare automată necesită un set de date care să poată furniza informații mai ofertante din punct de vedere a trăsăturilor unui semnal acustic. De aceea, procesul de extragere al caracteristicilor este unul esențial în cadrul proiectului.

Melspectrogramele au fost generate cu ajutorul funcției *melspectrogram* oferită de modulul *librosa*. Această funcție creează un vector 2D, în care fiecare valoare este reprezentată de energia unei benzi mel la un anumit moment de timp. Parametrii *n\_fft* și *hop\_length* ai funcției, setați cu valorile 2048 și respectiv 512 determină rezoluția Transformatei Fourier pe Termen Scurt, controlând modul în care semnalul este împărțit în cadre. Parametrul *n\_mels* specifică în câte benzi mel este împărțită axa frecvențelor, în acest caz fiind 128. Melspectrograma este convertită dintr-o scară liniară într-una logaritmică, exprimată în dB, folosind funcția *power.to.dB*, pentru a crea o reprezentare vizuală mai potrivită clasificării semnalelor.

Pentru a continua analiza semnalelor și din alte perspective, vom utiliza tot exemple pentru tastele *q* și *m,* pentru care s-a demonstrat anterior că arată diferențe și în analiza temporală și în cea spectrală. Examinând și reprezentări precum melspectrograme, MFCC sau coeficienți obținuți cu ajutorul Transformatei Wavelet se pot observa și alte informații care nu sunt la fel de vizibile în reprezentările anterioare.

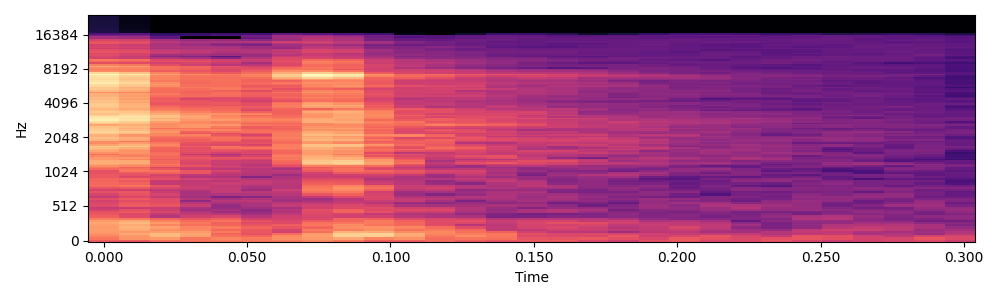


Figura 3.7 Melspectrograma pentru tasta „q”

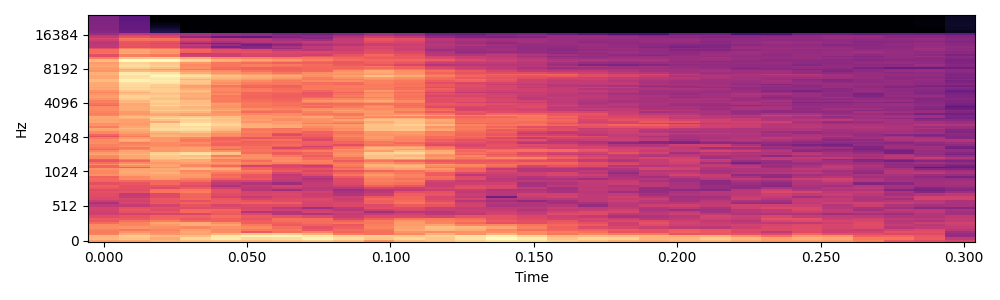


Figura 3.8 Melspectrograma pentru tasta „m”

În figura 3.7, corespondentă tastei q, se observă o concentrare a energie între 3000 și 8000 Hz, în special în intervalul 0-0,1 ms, care apoi scade, confirmând natura intensă și scurtă a semnalului. Figura 3.8 arată că energia este predominant localizată între 500 și 4000 Hz, aceasta fiind mai uniform distribuită și scade treptat, la fel ca în reprezentarea spectrală a tastei *m*. Melspectrogramele sunt reprezentări timp-frecvență, permițănd astfel ilustrarea ambelor caracteristici.

Coeficienții Mel-Cepstrali au fost extrași din semnalele audio prin intermediul funcției *mfcc* din cadrul pachetul *librosa*, care utilizează STFT cu dimensiunea unei ferestre de 2048 de eșantioane și distanța între ferestre de 512 de eșantioane. Pentru o reprezentare mai adecvată, este eliminat primul coeficient extras, deoarece aceasta prezintă o imagine globală a energiei semnalului și nu contribuie la diferențierea claselor, în timp ce restul coeficienților se axează pe informații mai relevante clasificării semnalelor acustice.

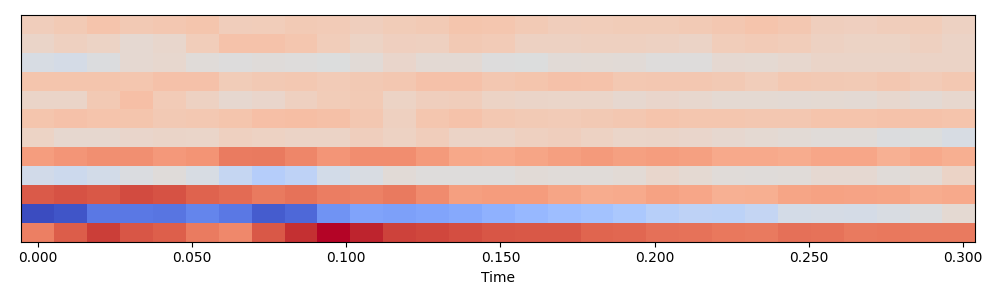


Figura 3.9 Reprezentarea MFCC a tastei „q”

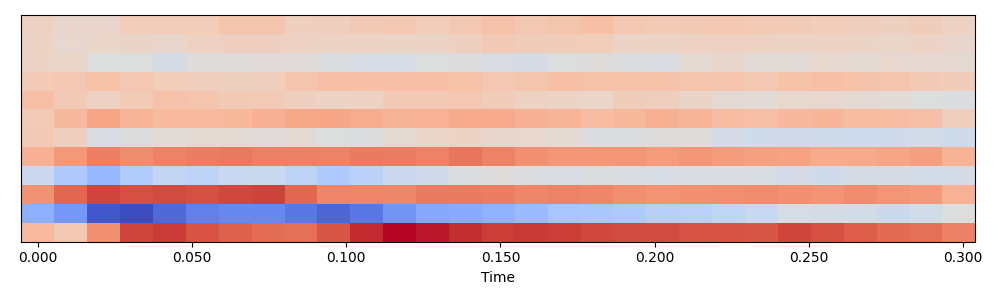


Figura 3.10 Reprezentarea MFCC a tastei „m”

În cadrul unei reprezentări a Coeficienților Mel-Cepstrali, axa Y reprezintă coeficienții, care ilustrează caracteristici diferite ale semnalelor. Primii doi coeficienți arată diferențele dintre frecvențele înalte și cele joase, în timp ce coeficienții 3-5 prezintă diferențele dintre un caracter brusc al semnalelor și unul mai rezonant. Coeficienții 6-8 ilustrează detalii care influențează spectrul, spre exemplu construcția tastaturii sau poziționarea microfonului. Ultimii coeficienți arată variațiile subtile sau zgomotul prezente în cadrul semnalului.Cu alte cuvinte, primii coeficienți se axează pe caracteristici generale, în timp ce coeficienții superior pun accent pe detaliile sunetului.

Figurile 3.9 și 3.10 confirmă diferențele dintre tastele *q* și *m* obținute anterior, însă adaugă și alte informații utile. Un aspect suplimentar al reprezentării MFCC este abilitatea de a evidenția evoluția profilului spectral în timp. Pentru tasta *q*, coeficienții arată variații mai abrupte, indicând faptul că profilul spectral al semnalului se schimbă rapid. Tasta *m* prezintă mai puține fluctuații ceea ce sugerează un semnal mai uniform și stabil. În plus, se poate observa că în cazul reprezentării pentru tasta *q* sunt activați mai mulți coeficienți în primele cadre, fiind un semnal cu o structură armonică mai complexă, spre deosebire de sunetul tastei *m* care arată un număr mai redus de coeficienți activați.

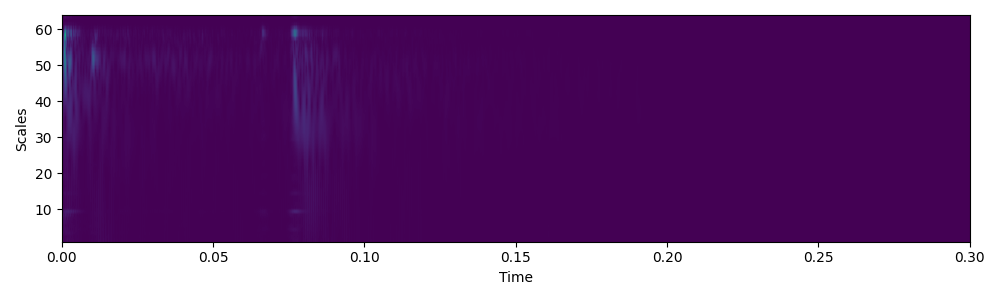


Figura 3.11 Reprezentarea Wavelet pentru tasta „q”

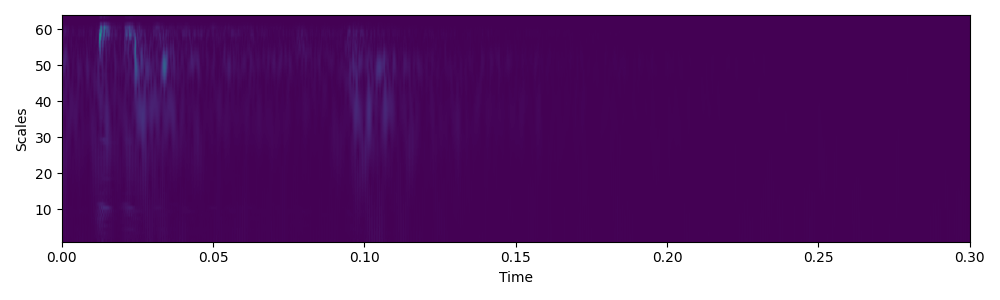


Figura 3.12 Reprezentarea Wavelet pentru tasta „m”

Pentru a putea calcula coeficienții Transformatei Wavelet, a fost definită funcția *wavelet\_transform.* În cadrul funcției, este folosită funcția predefinită *cwt* din utilitarul *pywt*, care are ca parametrii tipul funcției wavelet utilizat, în cazul proiectului fiind morlet, și o valoare pentru a determina rezoluția analizei wavelet. Funcția returnează o matrice alcătuită din valori complexe ale coeficienților, care ulterior sunt afișați sub forma unei imagini.

Reprezentările Wavelet oferă mai multe detalii din punct de vedere temporal, astfel evidențiind semnale tranzitorii. Figura 3.11 prezintă impulsuri de energie izolate, lucru care se putea observa și în reprezentări anterioare, dar cu o precizie temporală mai mică. Pentru tasta *m*, figura 3.12 descrie o reprezentare mai continuă, corespondent tranzițiilor mai fluide.

Inițial toate reprezentările realizate sunt de tip RGBA[[5]](#footnote-5), din acest motiv pentru a asigura compatibilitatea dintre acestea și modelele dezvoltate este necesară convertirea acestora în modul RGB. Convertirea imaginilor este realizată cu ajutorul funcției *convert* din cadrul modului *Imagine* importat din biblioteca *PIL*. Ulterior sunt salvate în format .png și sunt stocate corespunzător organizării din figura 3.2.

## 3.5 Arhitecturi și parametri

### 3.5.1 Rețeaua neuronală convoluțională (CNN)

Rețelele neuronale convoluționale reprezintă una dintre cele mai utilizate metode de clasificare a imaginilor datorită capacității lor de a identifica tipare și caracteristici în date bidimensionale. Din acest motiv, straturile utilizate în cadrul rețelei sunt de tip 2D.

Rețeaua conține 33 de straturi și este structurată astfel:

* 8 straturi de grupare
* 8 straturi de dezactivare a neuronilor[[6]](#footnote-6)
* 7 straturi convoluționale
* 7 straturi de normalizare[[7]](#footnote-7)
* 2 straturi dense
* 1 strat de aliniere[[8]](#footnote-8)

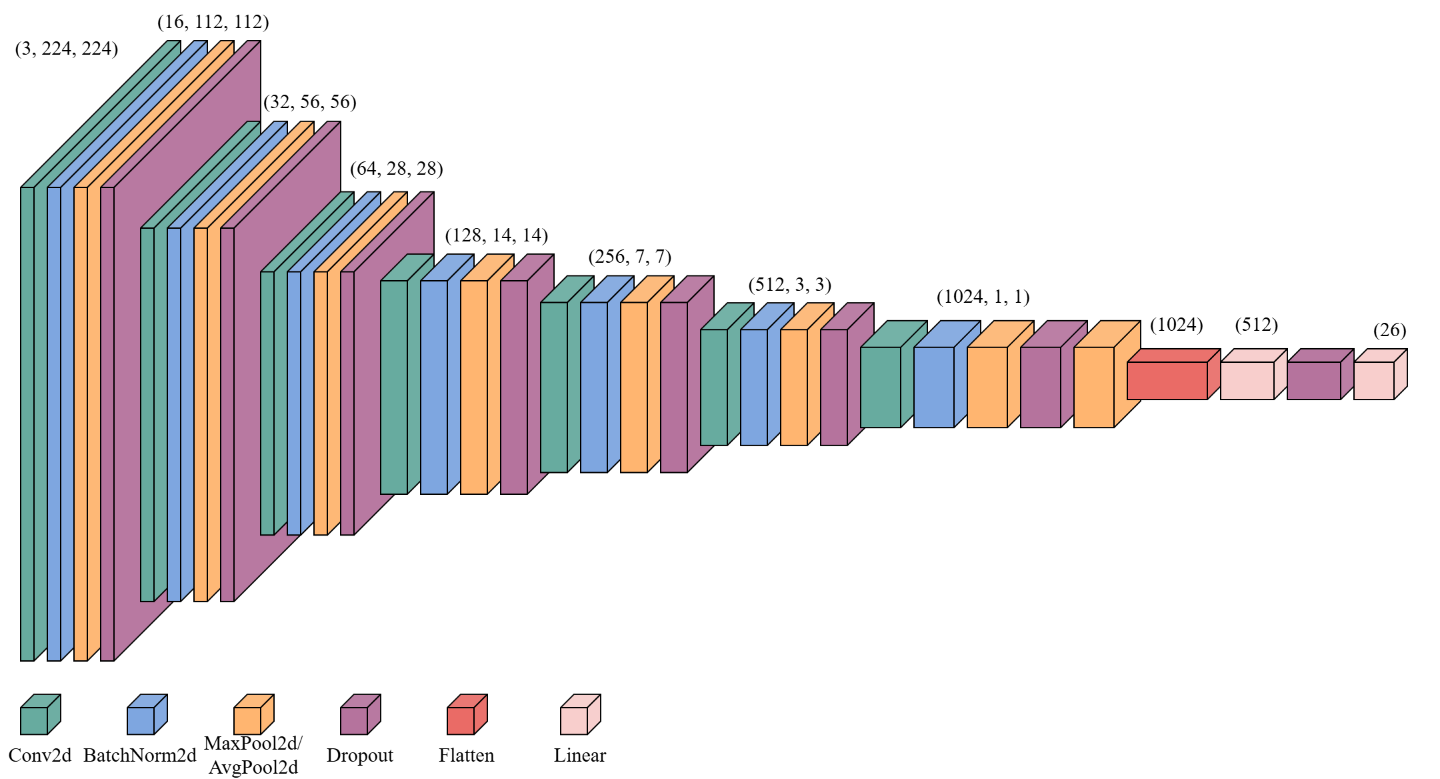


Figura 3.13 Structura CNN

Figura 3.13 nu prezintă doar structura rețelei, ci și evoluția dimensiunii datelor în urma trecerii prin straturile rețelei. Se observă o micșorare graduală a rezoluției imaginilor, un aspect care crește eficiența modelului. În același timp, numărul de canale crește, astfel permițând straturilor superioare să învețe caracteristicile definitorii pentru realizarea diferențierii între taste. Pentru a menține claritatea figurii și a nu adăuga detalii excesive, funcțiile de activare ReLU au fost omise, dar acestea sunt aplicate de-a lungul întregii arhitecturi, după fiecare strat de normalizare și primul strat dens. Ulterior, datele sunt transformate în format unidimensional pentru a putea fi transmis mai departe ultimului strat, și anumel cel de clasificare.

Fiecare strat are rolul său pe care îl îndeplinește cu ajutorul parametrilor care îl definesc. Prin ajustarea parametrilor, aspecte precum capacitatea de învățare, abilitatea de a generaliza și eficiența de calcul ale rețelei sunt puternic influențate. Parametrii sunt configurați astfel, cu mențiunea că cei neprecizați sunt setați cu valorile standard predefinite de Pytorch:

* Conv2d: *in\_features* = 3, 16, 32, 64, 128, 256, 512, *out\_features* = 16, 32, 64, 128, 256, 512, 1024, *kernel\_size* = 3, *stride* = 1, *padding* =1
* BatchNorm2d: *num\_features* = 16, 32, 64, 128, 256, 512, 1024
* MaxPool2d: *kernel\_size* = 2
* Dropout: *p* = 0,1, 0,2, 0,3, 0,4, 0,5
* Linear: *in\_features*=1024, 512, *out\_features* = 512, 26

### 3.5.2 CoAtNet

Combinând două tehnologii, CoAtNet este cunoscut pentu abilitățile sale de a identifica tipare și caracteristici în imagini complexe. Însă, pentru procesul lor de învățare, Transformers necesită un set de date foarte mare pentru a putea evita apariția supraînvățării. În plus, mecanismul „self-attention” implică faptul că pe măsură ce dimensiunea datelor de intrare se mărește, nevoia de memorie și puterea de calcul se mărește considerabil. Din acest motiv, pentru a asigura condițiile optime, sunt necesare echipamente hardware performante.

Prin urmare, pentru a preveni apariția acestor probleme, algoritmul CoAtNet implementat în cadrul proiectului presupune utilizarea doar blocurilor convoluționale. Algoritmul este open-source [55], la care am adus următoarele contribuții: înlocuirea blocurilor de tip Transformers cu blocuri de tip MBConv și reducerea numărului de blocuri prezente în fiecare stagiu cu scopul de a reduce complexitatea sistemului fără a pierde din performanța modelului.

|  |  |  |
| --- | --- | --- |
| Stagiu | Mărime | CoAtNet-0 |
| S0 – Conv | 1/2 | L=2 D=64 |
| S1 – MBConv | 1/4 | L=2 D=96 |
| S2 – MBConv | 1/8 | L=2 D=192 |
| S3 – MBConv | 1/16 | L=2 D=384 |
| S4 – MBConv | 1/32 | L=1 D=768 |

Tabel 1 Configurația arhitecturală a modelului CoAtNet-0

Configurația descrisă în Tabelul 1 presupune caracteristicile fiecărui stagiu al algoritmului, în acest caz fiind vorba doar de stagii convoluționale. Pentru fiecare stagiu, există o valoare care indică cu cât se reduc dimensiunile datelor și un anumit număr de blocuri repetate (L) și de canale de ieșire (D). CoAtNet-0 este cel mai de bază model inclus în familia modelelor CoAtNet, care demonstrează influența ajustării valorilor L și D.

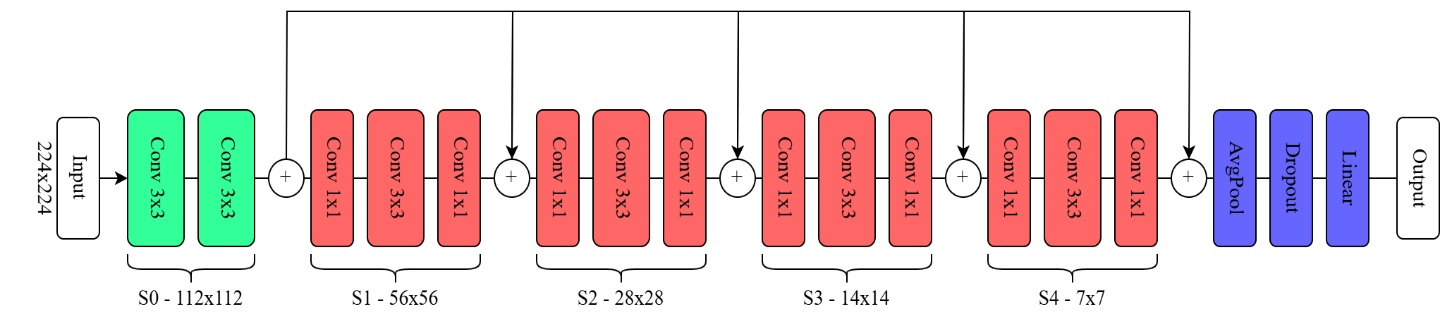


Figura 3.14 Structura utilizată pentru modelul CoAtNet

În figura 3.14 sunt prezentate stagiile algoritmului și ce implică acestea. În cadrul fiecărui bloc, există atât straturi de normalizare, de grupare și dezactivare a neuronilor, cât și funcții de activare pentru optimizarea modelului. De asemenea, fiecare stagiu include și conexiuni reziduale, fapt ce permite datelor de intrare să fie adăugate rezultatului ulterior.

Parametrii modelului sunt definiți într-un mod mai complex, deoarece algoritmul implică mai multe procese în cadrul arhitecturii sale. Primul bloc convoluțional are următorii parametri:

* Conv2d*: kernel\_size* = 3, *stride* = 1, 2, *padding* = 1, *bias* = False
* BatchNorm2d
* GELU

Exceptând primul bloc din fiecare stagiu, pentru blocurile MBConv, sunt setați parametrii expansion și downsample, cu valorile 4 și respectiv, downsample și au următoarea configurație:

* Conv2d: *kernel\_size* = 1, *stride* = 1, 2, *padding* = 0, *bias* = False
* BatchNorm2d
* GELU
* Conv2d: *kernel\_size* = 3, *stride* = 1, *padding* =1, *bias* = False (este setat și parametrul *groups* care este egal cu numărul de canal intermediare, acesta fiind stratul de convoluție „depthwise”)
* BatchNorm2d
* GELU
* Dropout: *p* = 0,2
* Conv2d: *kernel\_size* = 1, *stride* =1, *padding* = 0, *bias* = False

Primul strat convoluțional este stratul de expansiune și are rolul de a crește numărul de canale, astfel încât modelul să poate învăța caracteristici complexe fără a mări semnificativ numărul de parametri. Stratul de proiecție, ultimul strat, reduce numărul de canale pentru a comprima informația învățată în straturile de convoluție anterioare.

Atunci când parametrul downsample = True, există câteva schimbări structurale:

* MaxPool2d: *kernel\_size* = 3, *stride* = 2, *padding* = 1
* Conv2d: *kernel\_size* = 1, *stride* = 1, *padding* = 0, *bias* = False

Aceste modificări sunt utilizate pentru a asigura tranziția dintre rezoluții și a implementa o conexiune reziduală care să fie compatibilă din punct de vedere al dimensiunilor.

Dupa finalizarea blocurilor MBConv, există trei straturi finale:

* AdapativeAvgPool2d: *output\_size* = (1, 1)
* Dropout: *p* = 0,5
* Linear: *in\_features* = 768, *out\_features* = 26, *bias* = False

Numărul canalelor nu este setat manual, ci este definit cu ajutorul unei liste predefinită ale cărei valori se regăsesc în tabelul 1. Singura excepție este în cadrul blocurilor MBConv și presupune calcularea numărului de canale intermediare prin înmulțirea numărului de canale de intrare cu factorul de expansiune.

## 3.6 Etapa de antrenare

Procesul de învățare implică trei etape principale: etapa de antrenare, de validare și cea de testare. Etapa de antrenare reprezintă etapa în care modelul învăță prin ajustarea ponderilor, utilizând algoritmul descris în subcapitolul 2.2.1. La finalul fiecărei epoci de antrenare, este realizată etapa de validare în care nu mai sunt modificați parametrii rețelei, în schimb ea are ca scop evaluarea performanțelor modelului pentru a putea ajusta hiperparametrii. Un criteriu care arată că un model este performant este reprezentat de faptul că valorile rezultate după etapa de validare trebuie să fie cât mai apropiate de cele obținute în urma antrenării. Pentru a putea realiza comparații între mai multe scenarii, antrenarea a fost efectuată în aproximativ aceleași condiții.

Setul de date este împărțit, respectând următoarele proporții pentru fiecare etapă:

* 70% pentru etapa de antrenare
* 15% pentru etapa de validare
* 15% pentru etapa de testare

Pentru procesarea setului de date și împărțirea acestuia a fost implementată funcția *split*, în care imaginile sunt redimensionate la valorile 224x224 pentru a uniformiza datele de intrare, iar apoi sunt convertite în tensori, folosind modulul *transforms* oferit de biblioteca *torchvision*. Apoi datele trebuie etichetate și organizate în funcție de structura directoarelor, folosind utilitarul *ImageFolder* din modulul *datasets*. Setul de date este împărțit aleatoriu și cu o distribuție echilibrată a claselor cu ajutorul funcției *random\_split*, iar în final, utilitarul *DataLoader* asigură încărcarea datelor sub forma unor loturi („batches”). Pentru etapa de antrenare, imaginile din setul de date sunt amestecate pentru a evita învățarea mecanică. Numărul de loturi este un hiperparametru care se referă la numărul de exemple procesate în același timp într-o singură ajustare a parametrilor.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Numărul de epoci | Funcția de cost | Optimizatorul | Rata de învățare | Numărul de loturi |
| 10/50 | CrossEntropyLoss | Adam | 0,0001 | 8 |

Tabel 2 Parametrii de antrenare

Singurul parametru care diferă în configurațiile de antrenare pentru cele două modele este numărul de epoci. Acest lucru se datorează faptului că algoritmul CoAtNet, spre deosebire de CNN, obține o acuratețe ridicată în mai puține epoci, fapt ce face redundantă setarea unui număr mare de epoci..

Optimizatorul Adam are ca parametrii configurați rata de învățare și „weight\_decay”, setat la valoarea 0,0001. Setarea acestui parametru reprezintă o tehnică de regularizare pentru a evita apariția supraînvățării prin penalizarea valorilor mari ale ponderilor.

În timpul antrenării, sunt măsurați timpii necesari finalizării fiecărei epoci pentru a putea analiza eficiența modelelor, în final calculând durata totală a procesului de antrenare. Acest lucru este realizat folosind funcția *timer*, importată din modulul *timeit*. Timpii de antrenare diferă în funcție de complexitatea arhitecturii modelelor (numărul de straturi, funcții de activare, numărul de neuroni etc.) și de volumul datelor de intrare. În cadrul proiectului, durata procesului de învățare a fost:

* CNN-Melspectrograme: 43 de minute
* CNN-MFCC: 38 de minute
* CNN-Wavelet: 45 de minute
* CoAtNet-Melspectrograme: 35 de minute
* CoAtNet-MFCC: 40 de minute
* CoAtNet-Wavelet:37 de minute

Pentru a evalua calitatea procesului de antrenare, au fost calculate și reprezentate grafic acuratețea și pierderea, atât pentru etapa de antrenare, cât și pentru cea de validare. Metricile au fost reprezentate grafic prin implementarea a două funcții separate, *plot\_loss* și *plot\_accuracy*, utilizând biblioteca *matplotlib*.

## 3.7 Etapa de testare

După finalizarea etapelor de antrenare și validare, urmează etapa de testare care evaluează cât de bine generalizează modelul pentru date complet noi. Această etapă arată că îmbunătățirile apărute în cadrul primei părți a procesului de învățare există și într-un scenariu real. În timpul etapei de testare, modelul este setat în modul de evaluare, adică sunt dezactivate unele straturi (Dropout, BatchNorm2d etc.), astfel încât să nu existe instabilitate în cadrul rezultatelor.

În cadrul procesului de antrenare, este salvat modelul cu cea mai bună acuratețe de validare obținută pentru una dintre epoci, care este ulterior utilizat pentru etapa de test. Metricile folosite sunt acuratețea, matricea confuziilor, precizia și senzitivitatea, care sunt generate utilizând diferite funcții. Matricea confuziilor este obținută automat și reprezentată grafic cu ajutorul funcției *conf\_mat*, care utilizează biblioteci precum *torchmetrics* și *mlxtend*. Restul metricilor sunt calculate în funcția *print\_report*, folosind pachetul *sklearn.metrics*, care generează un raport ce conține toate datele și îl salvează local sub forma unui fișier .txt.

## 3.8 Generarea predicțiilor

Subcapitolul anterior prezintă etapa care face legătura între evaluarea modelului și utilizarea acestuia în cadrul unor aplicații reale. Un scenariu real implică realizarea unor înregistrări pentru care vor fi identificate și clasate tastele, astfel obținând o predicție.

Similar procesării datelor utilizate pentru antrenare și evaluare, procesul de generare a predicțiilor este realizat prin aplicarea aceluiași algoritm descris în subcapitolul 3.1. Reprezentările vizuale sunt obținute cu ajutorul funcției *generate\_image* și este foarte important ca acestea să corespundă din punct de vedere al aspectului cu cele generate pentru antrenare modelelor, deoarece și cele mai mici diferențe pot cauza erori în predicții.

Modelul determină, pentru fiecare literă segmentată, valori ale probabilităților pentru toate clasele posibile și afișează o listă cu primele șapte cele mai probabile litere. Cuvintele rezultate inițial sunt formate din literele asociate cu cea mai mare probabilitate, dar deoarece pot apărea erori, sunt generate mai multe cuvinte alternative. Acestea sunt obținute prin realizarea unor combinații posibile între literele existente în lista afișată anterior, iar pentru fiecare astfel de cuvânt este calculat un scor, înmulțind probabilitățile literelor care îl compun. Pentru a mări gradul de corectitudine al cuvântului este utilizată o listă predefinită de 50000 de cuvinte [56], cu ajutorul căreia sunt filtrate doar cuvintele existente. În final, rezultatul final este generat pe baza acestei liste și a celui mai mare scor obținut.

Pentru a prezenta rezultatele clar și intuitiv, a fost realizată o interfață web folosind biblioteca *Streamlit.* Interfața permite utilizatorilor încărcarea unui fișier .wav ce conține apăsări ale tastelor, iar apoi alegerea modelului dorit și a reprezentării vizuale. Odată ce aceastea au fost selectate, aplicația încărcă modelul corespunzător salvat local în timpul etapei de antrenare și realizează predicția cu ajutorul funcției descrise în paragraful anterior. Fișierul audio poate fi ascultat, butonul de pornire fiind afișat în urma încărcării înregistrării. Funcția *predic*t returnează predicția inițială, primele șapte predicții pentru fiecare segment audio, cuvintele alternative și predicția finală a sistemului, acestea fiind afișate și în cadrul interfeței.

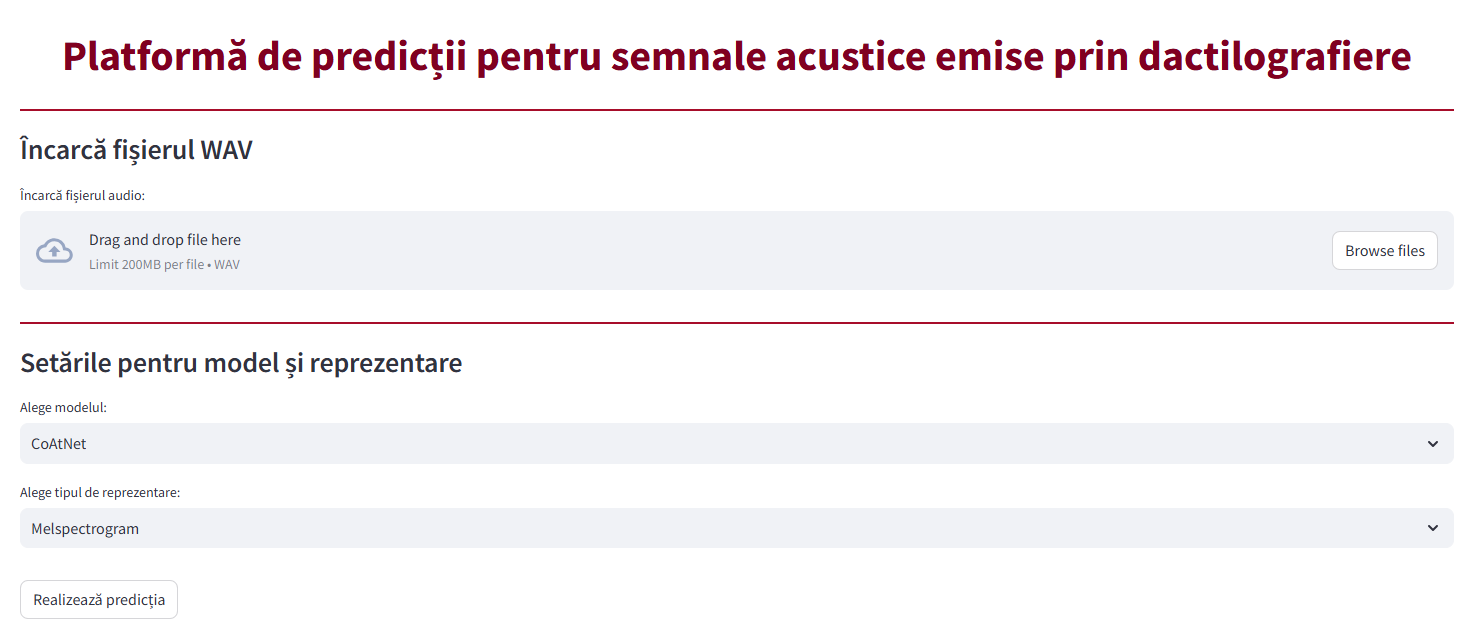


Figura 3.15 Interfața Web

# *Capitolul 4. Rezultate experimentale*

În această secțiune, sunt analizate mai multe cazuri, care includ antrenarea modelelor dezvoltate, utilizând diferite reprezentări vizuale. Capitolul are ca obiectiv analiza modului în care seturile de date diferite modifică rezultatele finale și procesul de învățare, prezentând o evoluție a curbelor de învățare, a matricii confuziilor și a restul metricilor asociate etapei de testare. În urma acestei analize, este realizată o comparație pentru a stabili cel mai performant scenariu. Din cadrul metricilor, focusul va fi asupra preciziei și a senzitivității, deoarece analiza comparativă a acestora oferă o imagine mai detaliată a performanței modelului, arătând câte dintre predicțiile efectuate au fost corecte, dar și capacitatea de a detecta toate aparițiile unei litere.

## 4.1 Performanțe CNN

### 4.1.1 Antrenarea pe baza melspectrogramelor

Primul scenariu este reprezentat de antrenarea rețelei neuronale convoluționale, având ca set de date melspectrograme. Procesul de antrenare este ilustrat în figura 4.1 în care sunt prezentate evoluțiile acurateței și ale funcției cost ale modelului, pe parcursul epocilor.

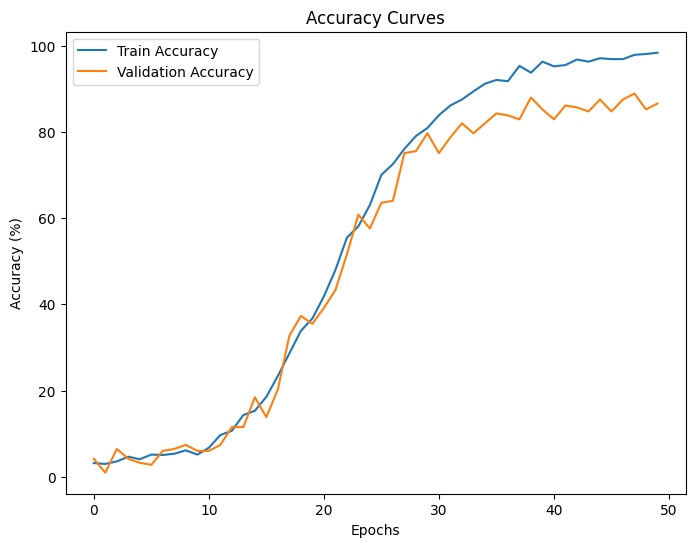
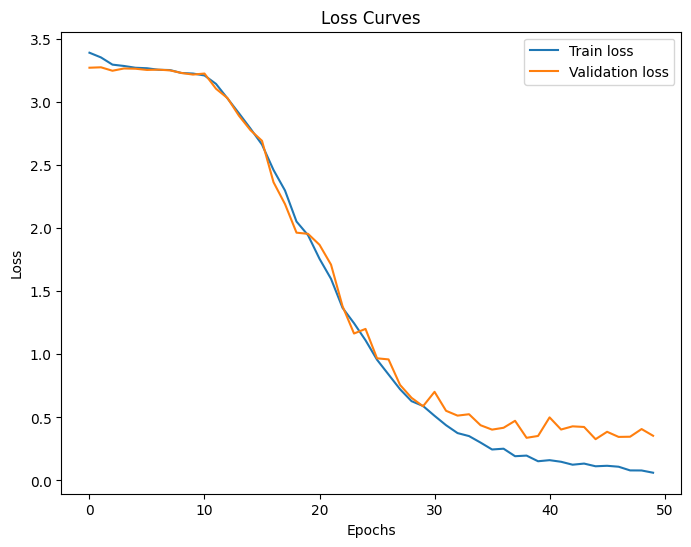


Figura 4.1 Curbele de învățare - CNN, Antrenarea pe baza melspectrogramelor

Atât în cadrul setului de antrenare, cât și pentru setul de validare, graficul pentru acuratețe arată o îmbunătățire semnificativă, evidențiind o creștere accentuată până în jurul epocii 30 și ulterior o perioadă de stabilizare a procesului. Analiza este continuată cu graficul funcției cost, care initial, are o valoare ridicată, spre finalul procesului de învățare scăzând aproape de valoarea 0, fapt ce ne arată că modelul învăță corespunzător și își reduce eroarea Din ambele grafice, observăm faptul că există o diferență între curba pentru antrenare și cea pentru validare. Cu toate că acest lucru sugerează o ușoară apariție a fenomenului de „overfitting”, adică o performanță mai ridicată pentru setul de antrenare făță de cel de validare, diferența este mică, deci modelul învață să generalizeze și pentru date noi. Apariția mai evidentă a supraînvățării a fost redusă prin adăugarea tehnicilor de regularizare și anume, straturile „dropout” și hiperparametrul „weight decay”.

În cadrul acestui proces, cele mai bune valori obținute pentru acuratețe sunt: 98, 75% pentru antrenare și 88,94% pentru validare.

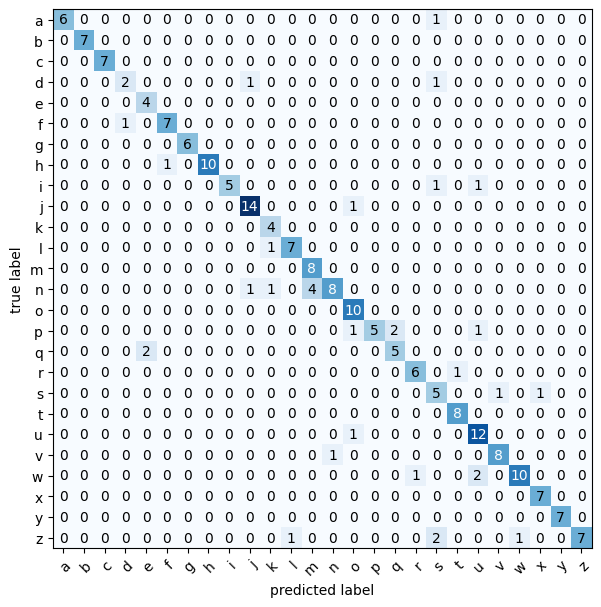


Figura 4.2 Matricea confuziilor - CNN, Antrenarea pe baza melspectrogramelor

În faza de test, utilizând modelul cu cea mai mare acuratețe pentru validare, s-a obținut valoarea 84,86% pentru acuratețe. Această diferență față de validare poate fi cauzată de aspecte precum: supraînvățare sau diferențe mici în setul de test (apariția zgomotului, stilul dactilografierii).

Figura 4.2 prezintă matricea confuziilor în care este evidențiată diagonala principală, demonstrând astfel faptul că majoritatea semnalelor au fost clasificate în mod corect. Există valori și înafara diagonalei pentru clase precum*: d*, *n*, *s* sau *z*, care au obținut și predicții false. Chiar dacă acestea sunt rare, ele pot evidenția cum unele similarități ale spectrogramelor sau fonetice pot influența rezultatele sistemului. Matricea confuziilor demonstrează că poziția tastelor joacă un rol important în predicțiile realizate, deoarece se pot observa clasificări greșite ale unor taste alăturate (tasta *a* a fost clasificată ca tasta *s*, tasta *n* a fost clasificată ca tasta *m* etc.) Acest lucru este cauzat de faptul că setul de date include apăsări ale tastelor realizate cu forță și unghi similar, ceea ce cauzează melspectrograme asemănătoare.

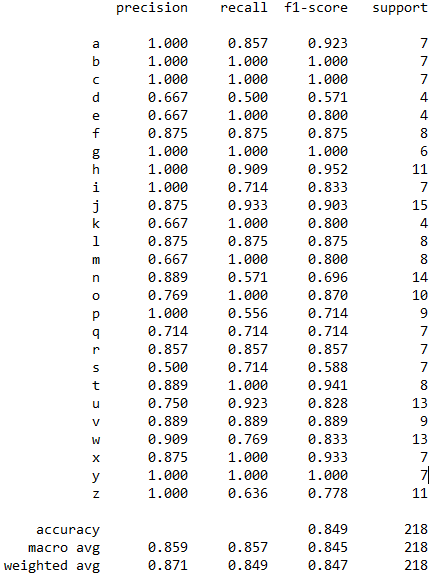


Figura 4.3 Metrici de evaluare - CNN, Antrenarea pe baza melspectrogramelor

Valorile preciziei, prezentate în figura 4.3, indică gradul de corectitudine pe care îl are modelul atunci când face predicții și în timp ce unele litere au obținut valori foarte mari (*a*, *b*, *c*, *h*, *j*, etc.), altele prezintă rezultate mai mici (*s*, *c*), cauzate de similirități sau mărimea limitată a setului de date. Comparativ, valorile senzitivității reflectă abilitatea modelului de a detecta corect toate exemplele corespunătoare unei clase, iar cele mai mici valori se gasesc în cazul claselor *m*, *n* și *z*. Diferența dintre cele două metrici arată că de obicei, modelul realizează predicțiile corect, dar poate identifica greșit unele exemple mai greu de recunoscut.

### 4.1.2 Antrenarea pe baza MFCC

În cazul antrenării pe baza MFCC, costul pentru setul de antrenare scade constant, spre deosebire de costul pentru setul de validare care în jurul epocii 20 stagnează și fluctuează în jurul valorii 1.

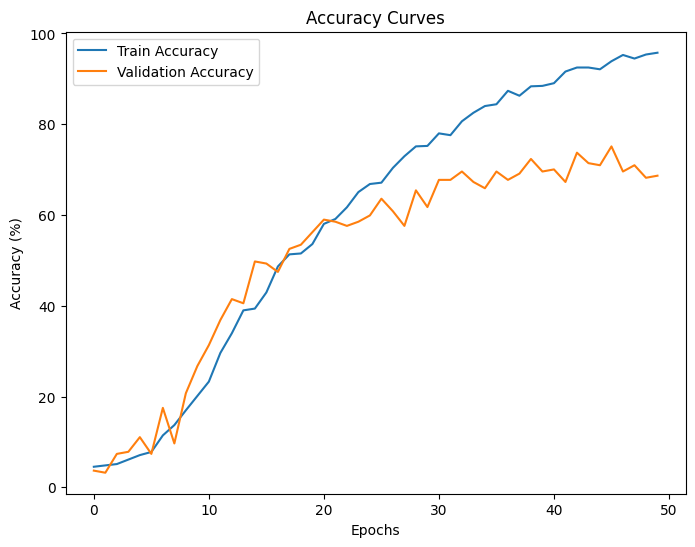
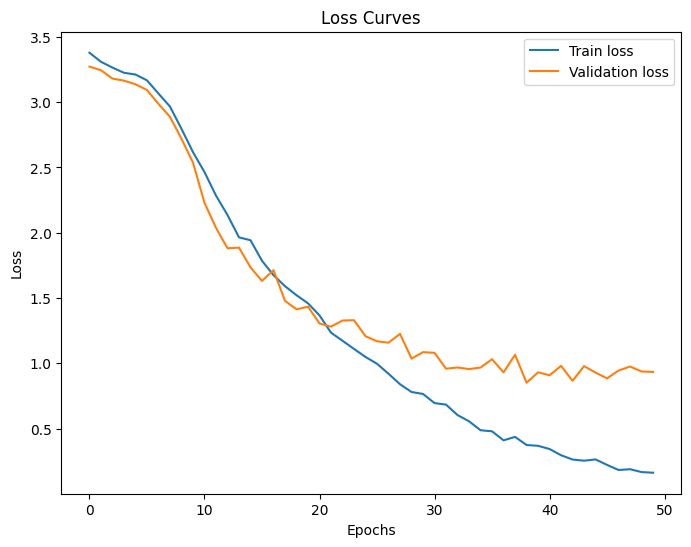


Figura 4.4 Curbele de învățare - CNN, Antrenarea pe baza MFCC

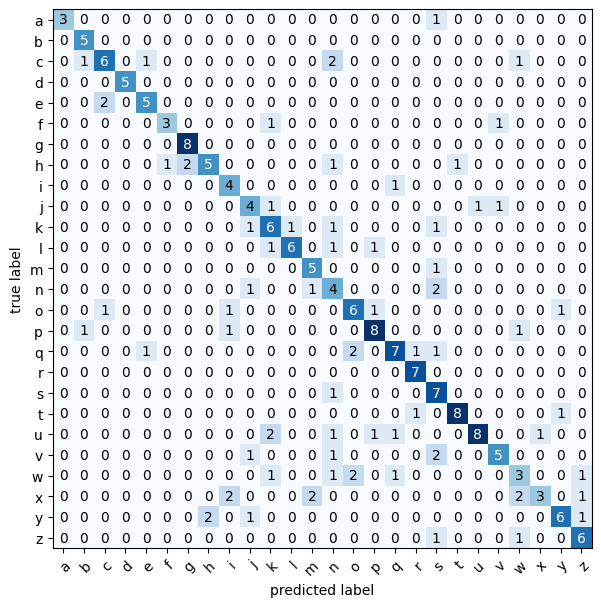


Figura 4.5 Matricea Confuziilor - CNN, Antrenarea pe baza MFCC

Acest lucru indică faptul că modelul continuă să se îmbunătățească pe baza datelor de antrenare fără a exista o astfel de evoluție și pentru date noi pentru model. Acuratețea obținută la finalul etapei de antrenare este 95,76%, în timp cea obținută în etapa de validare este 75,12%, demonstrând capacitatea limitată a modelului de a putea recunoaște tipare.

Acuratețea obținută în etapa de test este 65,60%, iar performanța este ilustrată în figura 4.5 prin intermediul matricei confuziilor, care deși prezintă o multitudine de valori pe diagonala principală, arată și un număr semnificativ de clasificări incorecte. Se observă că cel mai mare număr de predicții greșite a existat în cazul literelor *u* și *w*, pentru care s-au înregistrat 6 litere diferite, fapt ce indică că reprezentarea MFCC pentru aceste caractere nu oferă suficiente trăsături distinctive. Din punct de vedere al preciziei și senzitivității, cele mai slabe rezultate au fost obținute pentru literele *w*, *n*, *u*, *j*, demonstrând astfel că modelul realizează atât predicții incorecte pentru aceste litere, cât și identificarea greșită a acestora atunci când apar ca date de test.

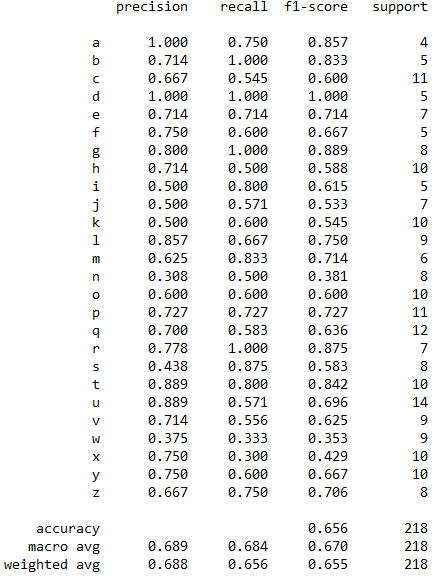


Figura 4.6 Metrici de evaluare - CNN, Antrenarea pe baza MFCC

### 4.1.3 Antrenarea pe baza Transformatei Wavelet

Curbele de învățare arată, și în acest caz, un proces de antrenare ce prezintă o evoluție consistentă dar este influențat de apariția supraîvățării. De data aceasta, din grafic se observă că diferența dintre cele două curbe apare mai devreme, în jurul epocii 15, arătând instabilitatea performanței pentru datele de validare, atât pentru cost, cât și pentru acuratețe.

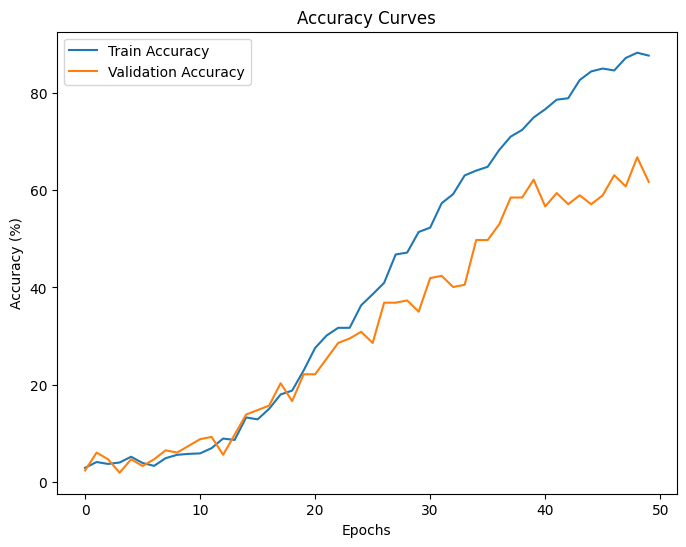
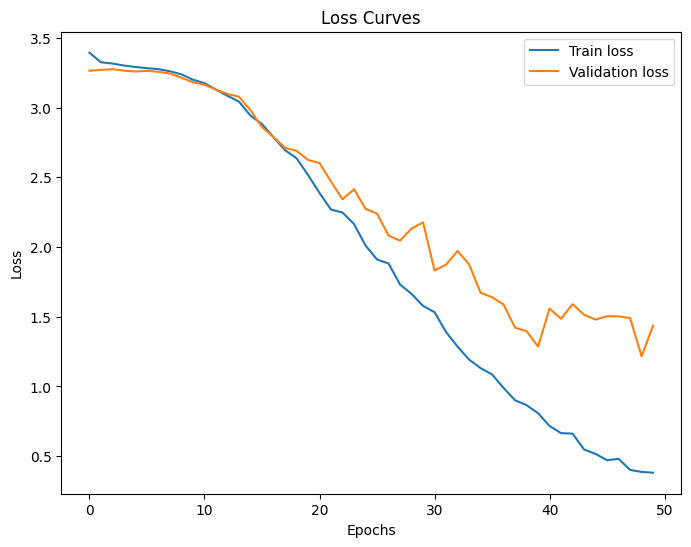


Figura 4.7 Curbele de învățare - CNN, Antrenarea pe baza Transformatei Wavelet

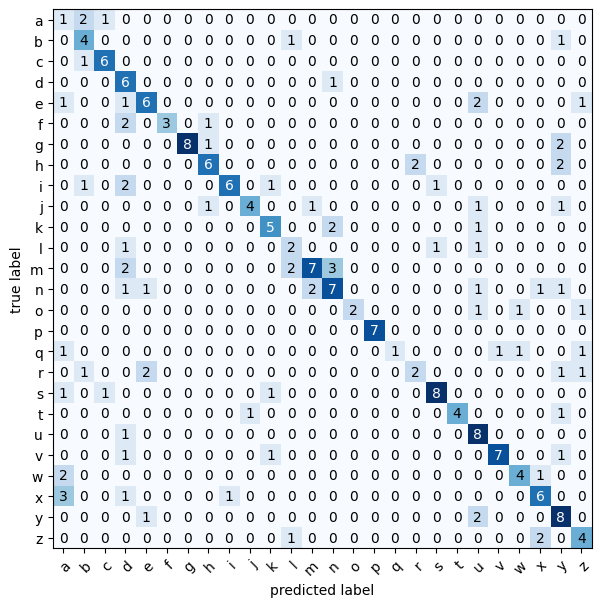


Figura 4.8 Matricea confuziilor - CNN, Antrenarea pe baza Transformatei Wavelet

Pentru etapa de antrenare, acuratețea obținută după finalizarea a 50 de epoci este 87,76%, pentru etapa de validare cea mai bună acuratețe este de 66,82%, iar pentru etapa de testare aceasta are valoarea de 60,55%.

Matricea confuziilor arată că pentru litera *r* s-a obținut cel mai mare număr de clasificări greșite, acestea fiind împărțite în mai multe clase, ceea ce indică faptul că modelul nu poate identifica corespunzător litera *r* pe baza reprezentării Wavelet. Din punct de vedere al preciziei, există o varietate de rezultate în funcție de clase. Pentru unele litere, cum ar fi: *f*, *o*, *p*, *q*, valoarea preciziei este foarte ridicată, lucru care de multe ori poate implica o valore scăzută a senzitivității (pentru *o* și *q*). Însă, există și rezultate mai scăzute ale preciziei, pentru literele *a* și *l*, evidențiind numărul ridicat de predicții greșite pentru aceasă clasă. Acest aspect este confirmat și de matricea confuziilor, din care se observă că predicțiile pentru *a* și *l* sunt distribuite incorect în mai multe clase,și pentru aceste litere fiind evidențiată limitarea modelui în a clasifica reprezentările.

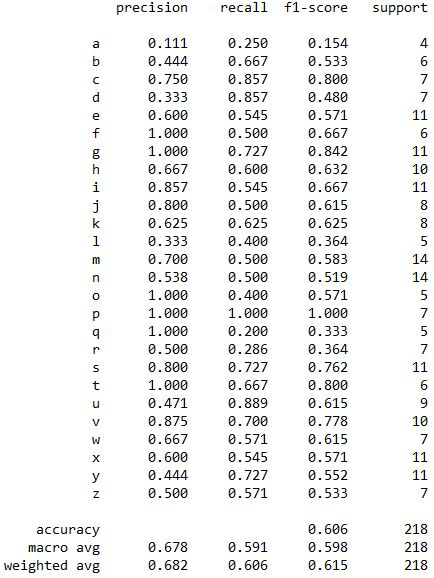


Figura 4.9 Metrici de evaluare - CNN, Antrenarea pe baza Transformatei Wavelet

## 4.2 Performanțe CoAtNet

### 4.2.1 Antrenarea pe baza melspectrogramelor

În cazul antrenării algoritmului CoAtNet pe baza melspectrogramelor, curbele de învățare sunt mult mai stabile în comparație cu cele generate în timpul antrenării rețelei convoluționale. Acest lucru poate avea mai multe cauze, printre care: arhitecturile diferite ale celor două modele, influența hiperparametrilor setați sau faptul că în acest caz, modelul a fost antrenat timp de 10 epoci, ceea ce implică mai puține ocazii de apariție a fluctuațiilor. Cu toate că numărul de epoci de antrenare este unul foarte mic, se observă din figura 4.11 că valorile pentru cost și acuratețe arată că modelul a reușit să obțină rezultate bune, atât pentru etapa de antrenare, cât și pentru cea de validare, încă din primele epoci.

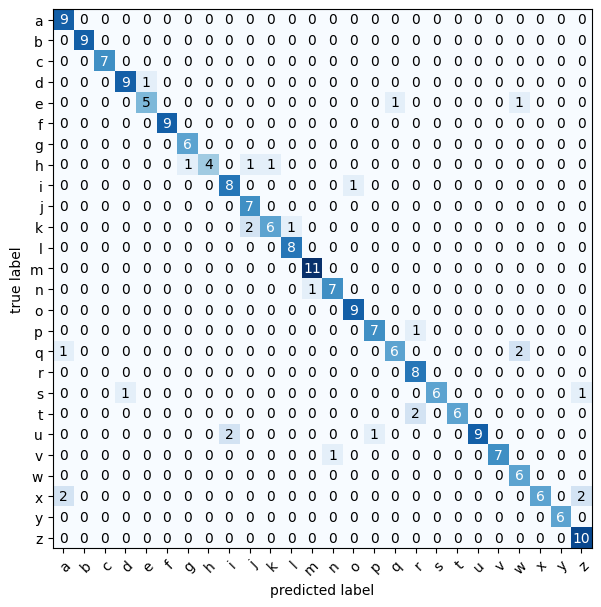


Figura 4.10 Matricea confuziilor - CoAtNet, Antrenarea pe baza melspectrogramelor

Pentru etapa de antrenare, acuratețea obținută este de 99,89%, pentru etapa de validare este de 84,79%, iar pentru etapa de test este de 87,61%, demonstrând capacitatea de generalizare a modelului și faptul că modelul nu a suferit de o apariție semnificativă a supraînvățării, în ciuda valorii foarte ridicate atinse pentru datele de antrenare.

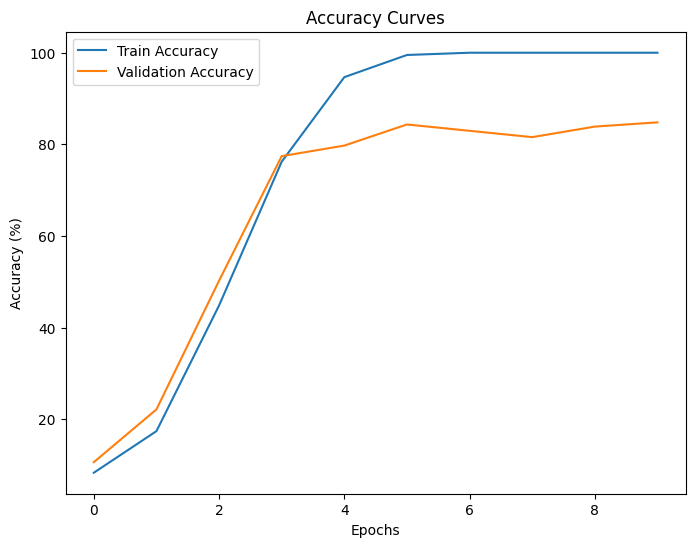
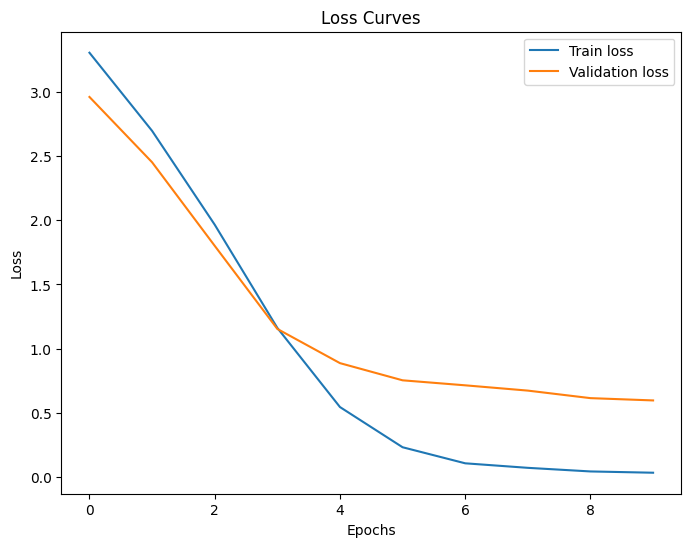


Figura 4.11 Curbele de învățare - CoAtNet, Antrenarea pe baza melspectrogramelor

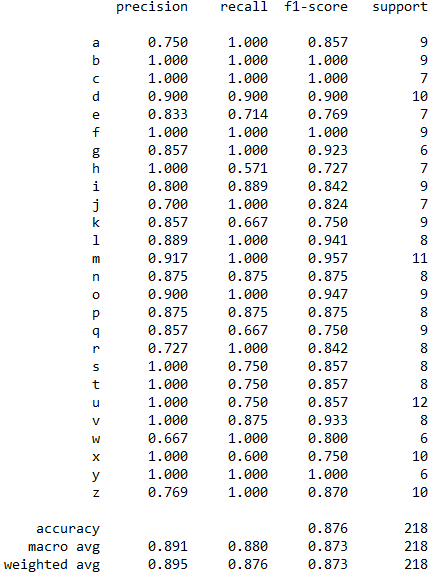


Figura 4.12 Metrici de evaluare - CoAtNet, Antrenarea pe baza melspectrogramelor

Matricea confuziilor evidențiază din nou acest aspect, prin faptul că prezintă predicții corecte pentru majoritatea claselor, cu câteva excepții pentru litere ca *x*, *h* sau *q*, pentru care s-au înregistrat cele mai multe clasificări greșite. Se observă, din figura 4.12, că pentru aceleași litere, au fost obținute cele mai scăzute valori ale senzitivității. Pentru precizie, în cazul a 17 litere din 26, s-au înregistrat valori foarte ridicate, cel mai mic rezultat fiind pentru litera w, care a obținut un număr mai mare de valori fals pozitive.

### 4.2.2 Antrenarea pe baza MFCC

În acest caz, din curbele de învățare se observă o apariție mai semnificativă a fenomenului de „overfitting”, funcțiile cost pentru ambele etape scăzând, dar doar cea pentru antrenare coboară constant și se apropie de valoarea 0. Graficul pentru acuratețe, de asemenea arată faptul că în cazul etapei de antrenare, aceasta atinge valoarea de 99,70%, indicând practic o memorare a datelor, în timp ce pentru etapa de validare obține valoarea maximă 72,81% la epoca 7.

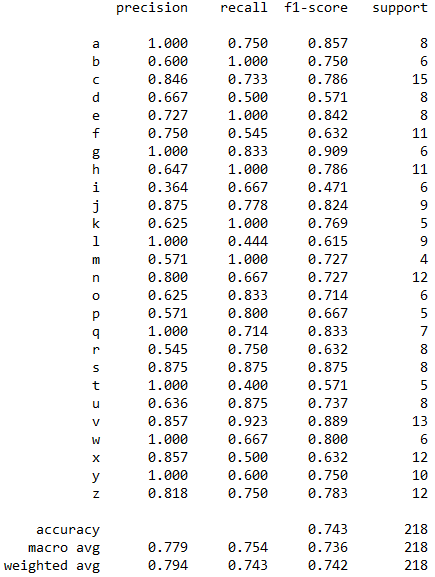


Figura 4.13 Metrici de evaluare - CoAtNet, Antrenarea pe baza MFCC

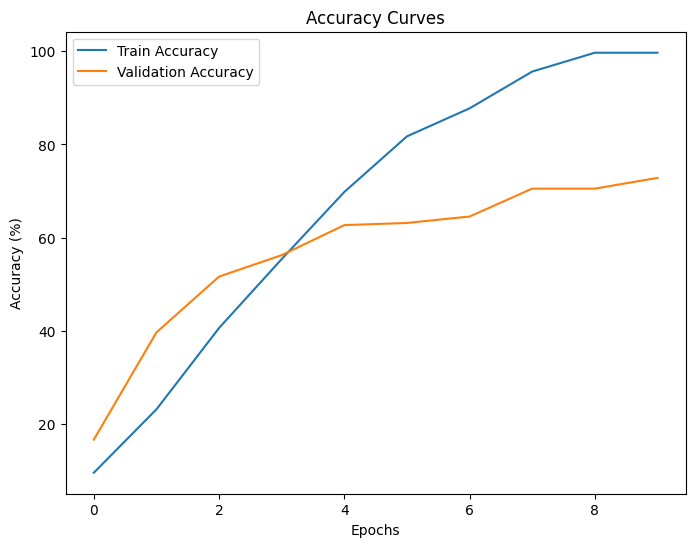
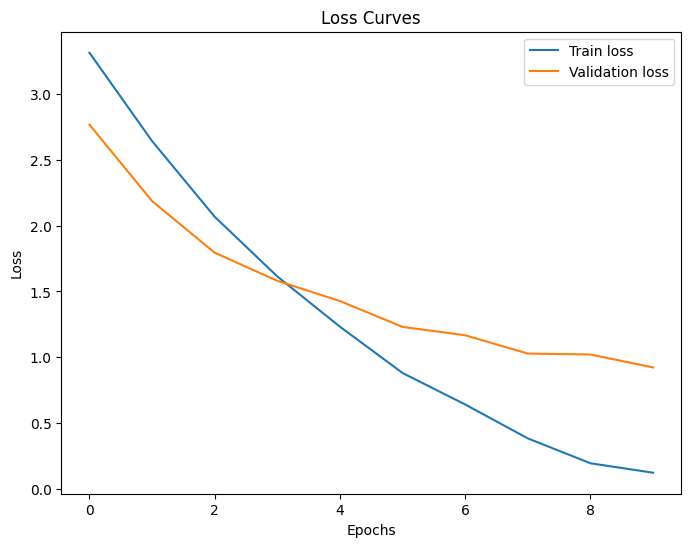


Figura 4.14 Curbele de învățare - CoAtNet, Antrenarea pe baza MFCC

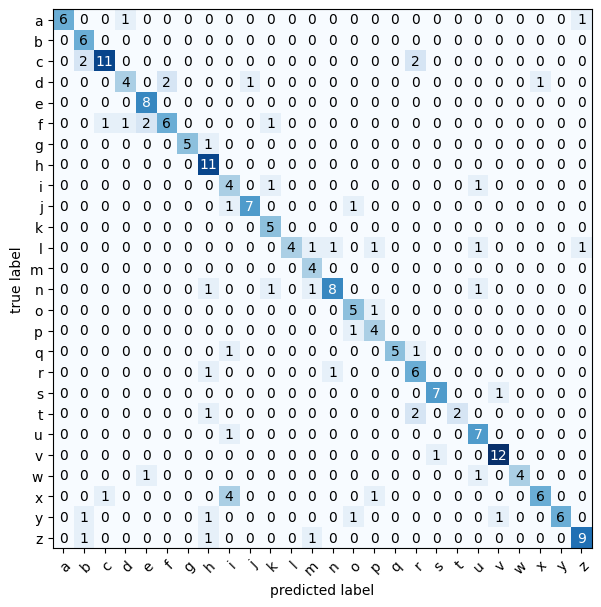


Figura 4.15 Matricea confuziilor - CoAtNet, Antrenarea pe baza MFCC

Rezultatele obținute în urma etapei de test indică o performanță medie a modelului, cu o acuratețe de 74,31%. Figura 4.14 ilustrează matricea confuziilor, care și în acest caz arată o prezență majoritară a valorilor în cadrul diagonalei principale, dar care evidențiază și limitări în clasificarea anumitor litere, precum: *x*, pentru care a existat cel mai mare număr de predicții incorecte fiind clasificate ca litera *i* sau *t*, pentru care s-au înregistrat 2 predicții corecte și 3 greșite.

Din perspectiva altor metrici, prezentate în figura 4.15, pentru precizie, rezultatele variază considerabil, având valori mari pentru litere precum: *a*, *l* sau *t* și valori scăzute cuprinse între 30% și 50% pentru literele *i* și *m*. În ciuda valorilor ridicate ale preciziei, pentru literele *l* și *t* au fost obținute cele mai scăzute valori ale senzitivității, modelul nereușind să identifice corespunzător toate aparițiile acestor clase. Diferența dintre rezultatele obținute pentru cele două metrici indică faptul că modelul preferă să nu realizeze predicții atunci când nu este foarte sigur.

### 4.2.3 Antrenarea pe baza Transformatei Wavelet

Ultimul scenariu este reprezentat de antrenarea algoritmului CoAtNet pe baza reprezentării Wavelet, un proces care implică o scădere mai lentă a costului pentru ambele etape, însă începând cu epoca 5, conform figurei 4.16, spre deoesbire de costul pentru datele de antrenare care continuă să scadă, costul pentru datele de validare începe să stagneze și să fluctueze, indicând și în acest caz apariția supraînvățării. În cadrul etapei de antrenare, modelul atinge acuratețea de 97,73%, iar în timpul etapei de validare este obținută valoarea considerabil mai redusă de 69,12%, aceeastă diferență reflectând același comportament ca în cadrul evoluției costului.

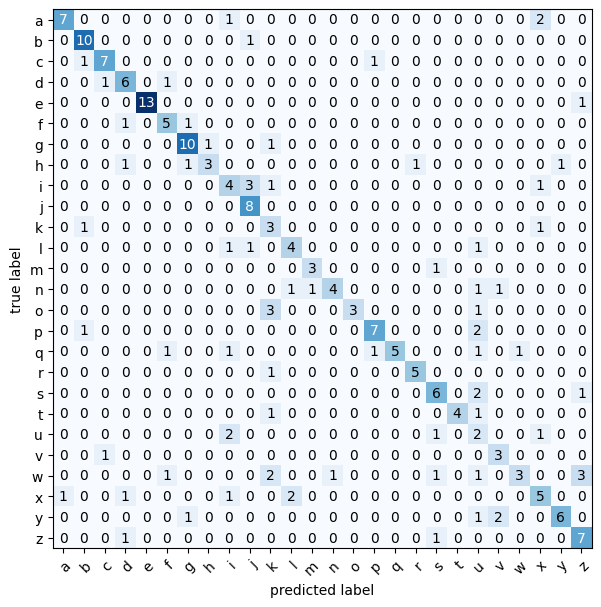


Figura 4.16 Matricea confuziilor - CoAtNet, Antrenarea pe baza Transformatei Wavelet

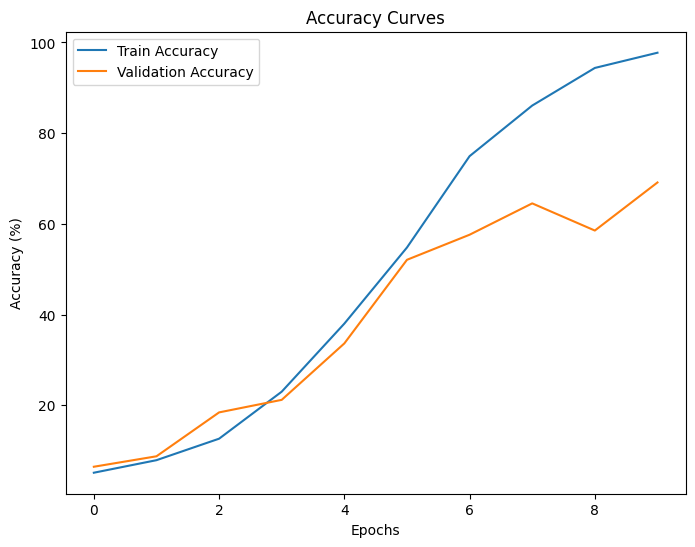
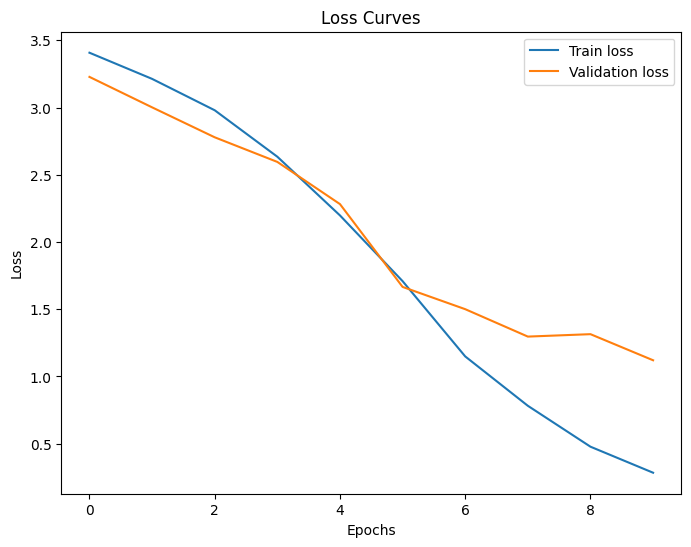


Figura 4.17 Curbele de învățare - CoAtNet, Antrenarea pe baza Transformatei Wavelet

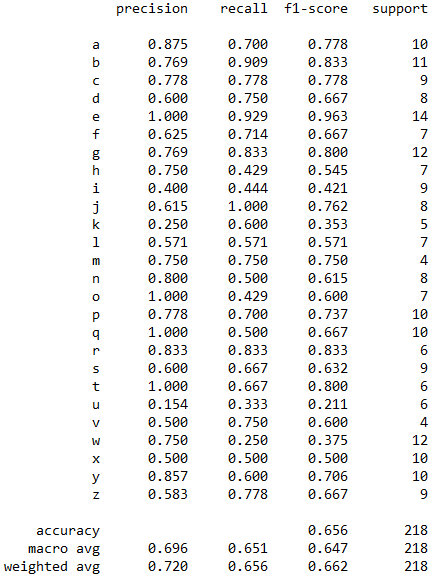


Figura 4.18 Metrici de evaluare - CoAtNet, Antrenarea pe baza Transformatei Wavelet

Acuratețea obținută la finalul etapei de test este de 65,6%, indicând că modelul reușește să clasifice corect puțin peste jumătate din exemple. Figura 4.17 prezintă matricea confuziilor, ce ilustrează clase cu majoritatea predicțiilor corecte, precum *b*, *e* sau *g*, dar și litere pentru care au fost înregistrate mai multe clasificări incorecte, de exemplu litera *w*, care a fost clasificată corect de 3 ori și greșit de 8 ori.

Valorile variază și în cadrul preciziei și a senzitivității, obținând rezultate bune pentru precizie pentru anumite litere, cum ar fi *e*, *o*, *q* sau *t*, dar și rezultate la o diferență considerabilă mai mici pentru *u* și *k*. Similar, senzitivitatea înregistrează valori variabile de la 92,9% pentru *j* până la 25% pentru *w*.

## 4.3 Analiză comparativă

Pentru a oferi o imagine globală asupra performanțelor obținute, este necesară o analiză simultană și comparativă a scenariilor prezentate anterior, fiecare evidențiind diferențe între performanță și capacitatea de a generaliza.

Melspectrogramele au reprezentat cel mai bun suport pentru a antrena modelele din cadrul proiectului, cu mențiunea că algoritmul CoAtNet a reușit să obțină rezultate puțin peste cele obținute de CNN. În urma utilizării MFCC ca set de date, s-a observat o scădere în cadrul rezultatelor pentru ambele modele și o apariție mai pronunțată a supraînvățării, reprezentările nefiind suficient de relevante pentru a se putea realiza corect clasificarea literelor. Un motiv posibil al acestor rezultate este că reprezentările Coeficienților Mel-Cepstrali sunt versiuni comprimate prin reducerea axei frecvenței la un număr limitat de coeficienți, altfel rezultând într-o reprezentare ce nu poate ilustra toate detaliile necesare. Deși există mai multe studii care utilizează MFCC pentru a rezolva cerința proiectului, aceștia sunt folosiți ca valori, astfel utilizarea lor sub forma unor imagini pentru a antrena modelele nereușind să obțină cele mai bune rezultate.

Cele mai nesatisfăcătoare rezultate au fost obținute în urma antrenării modelelor folosind Transformata Wavelet, observându-se o scădere a stabilității și a capacității de generalizare. În capitolul 3, a fost remarcat faptul că, spre deosebire de melspectrograme sau reprezentările Coeficienților Mel-Ceptrali, cele bazate pe Transformata Wavelet ilustrează doar câteva porțiuni mai evidențiate și foarte multe zone întunecate. Ca urmare, deși această abordare poate fi utilă în analiza semnalelor tranzitorii, structura vizuală a Transformatei Wavelet nu oferă o cantitate suficientă de date pentru a putea realiza clasificarea tastelor.

În toate cele 3 cazuri, CoAtNet a reușit să atingă performanțe mai bune și stabile, lucru care poate fi datorat arhitecturii algoritmului, mai complexă decât cea a CNN, permițând astfel extragerea unor trăsături mai relevante procesului de clasificare. În plus, integrarea convoluției „depth-wise” și a conexiunilor reziduale a contribuit la creșterea stabilității procesului de actualizare a parametrilor.

## 4.4 Influența zgomotului

Cum antrenarea modelelor a fost realizată pe baza unor înregistrări realizate într-un mediu fără zgomot, este de interes observarea consecințelor introducerii zgomotului în cadrul datelor de test. Am utilizat modelul cu cea mai bună performanță, mai exact CoAtNet, și tipul de reprezentare ce a obținut cele mai favorabile rezultate, adică melspectrogramele. Inițial, am antrenat modelul folosind aceleași date ca în cazurile anterioare, fără zgomot, iar pentru etapa de test am modificat setul de date pentru a conține doar exemple cu zgomot. Exemplele pentru acest caz au fost realizate prin înregistrarea sunetelor tastelor împreună cu alte perturbări din mediul încojurător, astfel simulând condiții mai realiste.

Se observă din figura 4.19 că modelul învață corespunzător, păstrând comportamentul anterior și valori ridicate pentru acuratețe, 99,65% pentru datele de antrenare și 87,15% pentru datele de validare. Însă etapa de test demonstrează sensibilitatea modelului la zgomot, reușind să clasifice corect doar 32,18% dintre exemple.

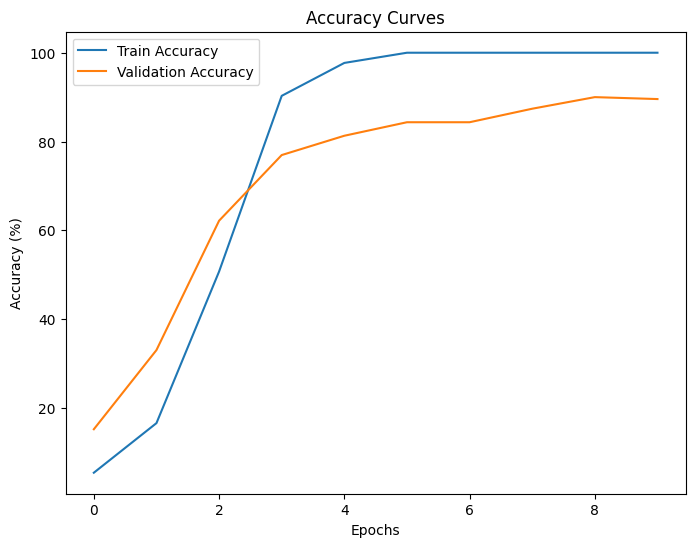
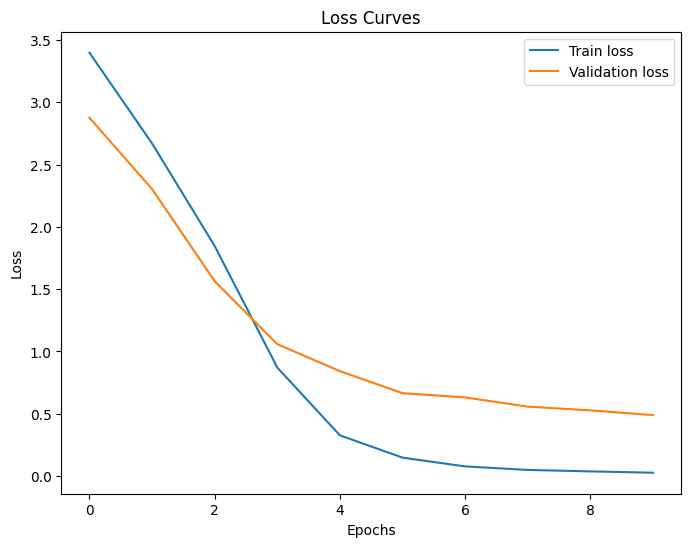


Figura 4.19 Curbele de învățare după inserarea zgomotului

Acest lucru este ilustrat în figura 4.20, în care matricea confuziilor evidențiază faptul că deși există câteva clase pentru care rezultatele au fost mai favorabile cum ar fi *g* și *i*, majoritatea claselor au înregistrat o multitudine de predicții incorecte, litere precum *d*, *j* sau *w* având 0 clasificări corecte.

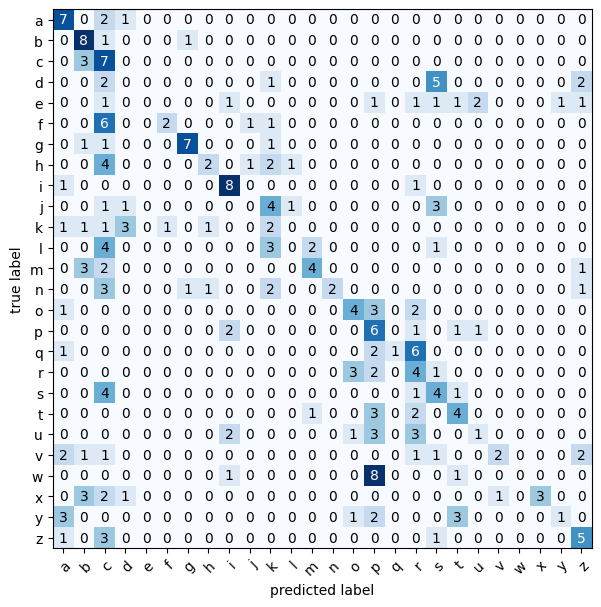


Figura 4.20 Matricea confuziilor după inserarea zgomotului

Suportul de date din acest caz este fixat la valoarea 10, deoarece datele de test nu au mai fost selectate în mod aleator. Directorul de test, ce conține câte 10 exemple pentru fiecare clasă, fiind separat intenționat pentru a nu introduce zgomotul în etapele de antrenare și validare. De asemenea, și restul metricilor prezintă valori scăzute pentru majoritatea literelor, sugerând că nu doar acuratețea este afectată, dar și capacitatea modelului de a face predicții corecte atunci când întâlnește o anumită literă, respectiv de a recunoaște toate aparițiile acelei litere.

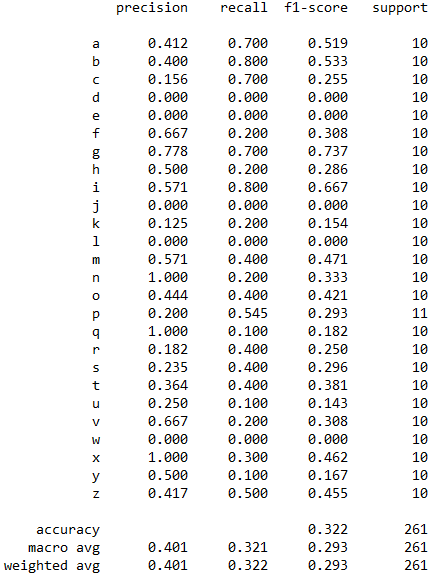


Figura 4.21 Metrici de evaluare după inserarea zgomotului

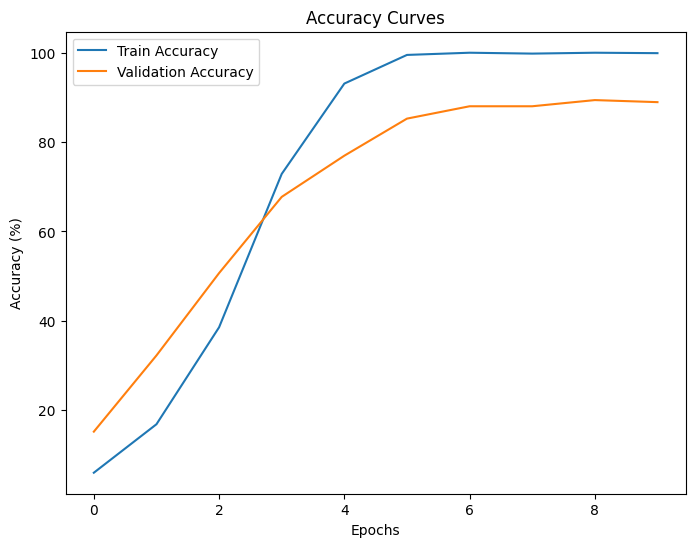
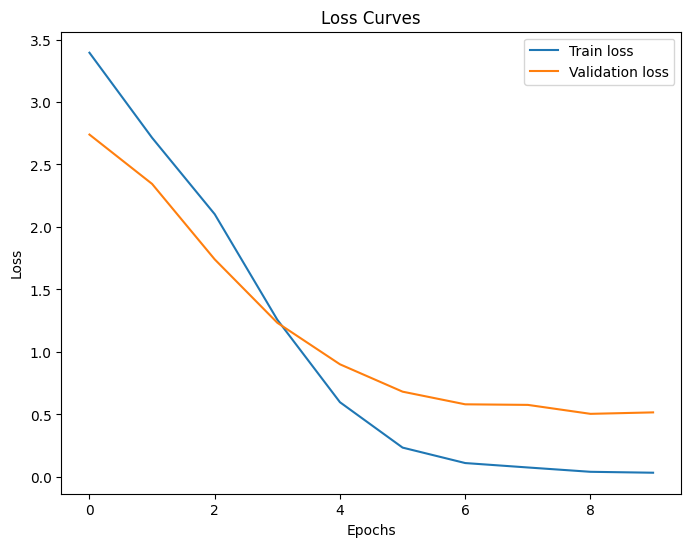


Figura 4.22 Curbele de învățare după inserarea zgomotului în cadrul tutror etapelor

Ulterior, pentru a continua analiza, am realizat antrenarea modelului adăugând date ce conțin zgomot în cadrul tuturor etapelor pentru a observa gradul de îmbunătățire al rezultatului obținut în scenariul de mai sus.

Pentru primele două etape, curbele de învățare arată că rezultatele nu se schimbă semnificativ și rămân aproximativ egale cu cele din scenariul anterior. Acest caz atinge cea mai ridicată valoare pentru acuratețea obținută în timpul etapei de validare, și anume 89,4%, lucru care poate fi datorat introducerii zgomotului care poate duce la îmbunătățirea capacității de generalizare a modelului și diminuarea efectelor cauzate de supraînvățare.

Etapa de test obține, în acest caz, o acuratețe de 85,78% ceea ce demonstrează o îmbunătățire semnificativă în comparație cu cea obținută antrenând fără exemple ce conțin zgomot. Aceste rezultate se reflectă și în matricea confuziilor, ilustrată în figura 4.23, dar și în restul metricilor, precum precizia sau senzitivitatea. În acest scenariu, cele mai nefavorabile rezultate au fost obținute pentru litera *x*, care a înregistrat 5 clasificări greșite, astfel obținând o valoare redusă a senzitivității.

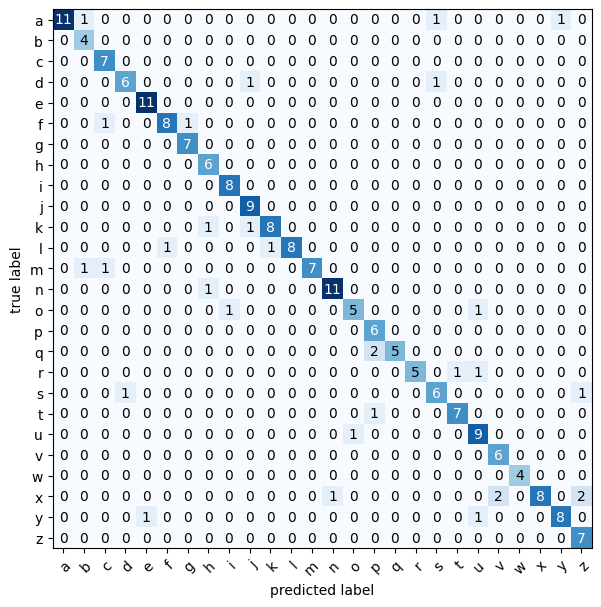


Figura 4.23 Matricea confuziilor după inserarea zgomotului în cadrul tuturor etapelor

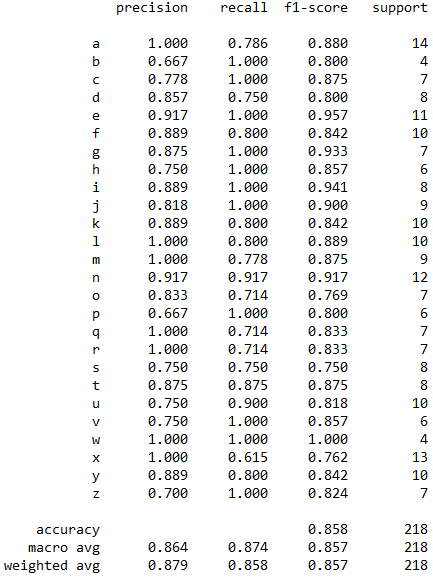


Figura 4.24 Metrici de evaluare după inserarea zgomotului în cadrul tuturor etapelor

## 4.5 Obținerea predicțiilor

Pentru a observa rezultatele obținute în urma generării predicțiilor, am ales cuvântul „exemplu” pentru a analiza mai multe cazuri, evidențiind capacitatea modelelor de a identifica tastele în condiții diferite.

Prima dată, am realizat înregistrarea tastând rar fiecare caracter și în condiții identice ca cele în care s-au înregistrat tastele pentru antrenarea modelului. Deoarece scenariul bazat pe CoAtNet și melspectrograme a obținut cele mai bune performanțe, acesta este utilizat pentru a realiza predicțiile.

După finalizarea înregistrării, sunt respectați aceeași pași descriși în capitolul anterior, ulterior aceasta fiind încărcată în cadrul aplicației. Se observă în figura 4.25 că predicția inițială a modelului este „pximpju”, o secvență ce conține trei litere clasificate incorect din șapte. Primele șapte predicții oferite de model evidențiază faptul că cele 3 litere se regăsesc și pot fi ulterior utilizate pentru a realiza o predicție finală și corectă. În final, este returnat cuvântul dorit, după înlocuirea caracterelor incorecte prin algoritmul descris în subcapitolul 3.8.



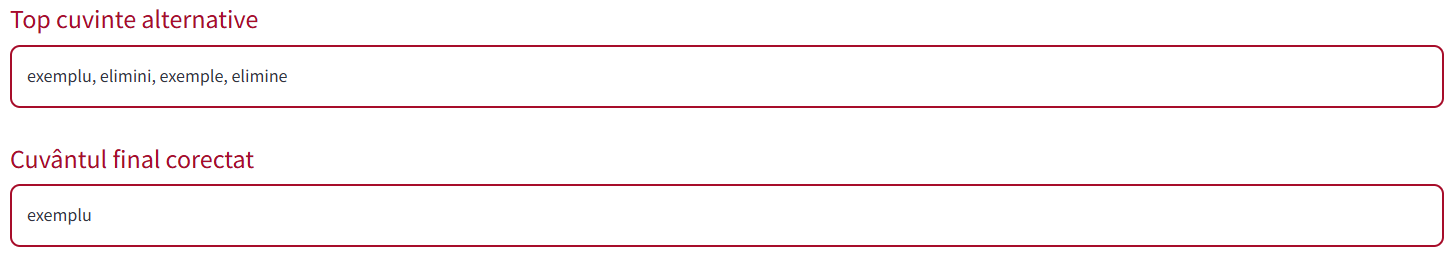


Figura 4.25 Realizarea predicțiilor - Cazul I

Al doilea caz analizat este cel în care tastele sunt apăsate mai des pentru a putea evidenția un factor principal care poate influența predicțiile obținute și anume viteza de dactilografiere.

Figura 4.26 arată că inițial cuvântul obținut conține 4 predicții greșite pentru literele *e*, *l* și *u*, aspect similar cu cel din scenariul prezentat anterior. Se remarcă faptul că pentru litera 5, primele șapte predicții nu oferă litera l, astfel algoritmul nereușind să identifice corespunzător toate literele necesare pentru a putea avea un rezultat final corect. Acest lucru poate fi cauzat de faptul că prin apăsarea mai frecventă a tastelor există o probabilitate mai ridicată de apariție a suprapunerilor dintre taste, conducând la greșeli în procesul de segmentare a literelor și ulterior la o clasificare incorectă.

Totuși, se observă asemănări între seturile de predicție oferite pentru fiecare literă, ceea ce demonstrează o stabilitate a modelului în opțiunile de clasificare. De exemplu, în ambele cazuri litera *e* este asociată cu litere precum *p*, *u* sau *t*, iar pentru litera *l*, lista predicțiilor conține *j*, *k* sau *n*.



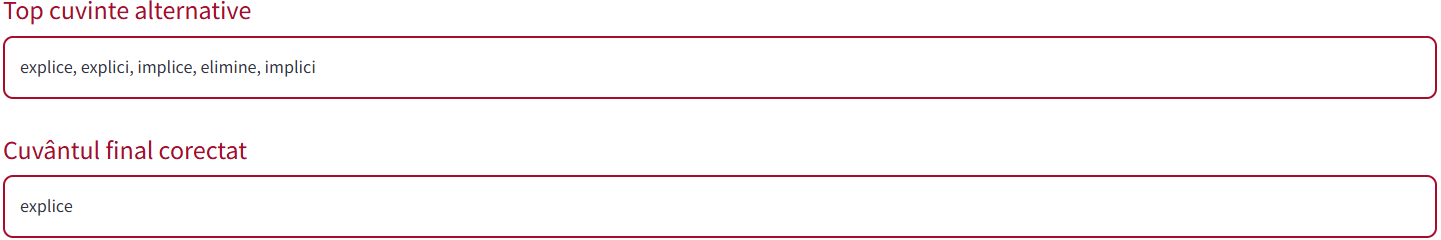


Figura 4.26 Realizarea predicțiilor - Cazul II

# *Capitolul 5. Concluzii și direcții viitoare*

Lucrarea a avut ca scop principal studiul și dezvoltarea unui sistem capabil să identifice și să claseze semnalele acustice emise prin dactilografiere, începând cu o descriere teoretică a semnalelor acustice acustice și a algoritmilor, și urmând implementarea practică a acestor noțiuni. O parte importantă a proiectului constă în analiza rezultatelor obținute în urma antrenării și evaluării modelelor în diferite scenarii, deoarece acestea oferă cele mai multe informații cu privire la utilizarea sistemului într-o aplicație practică. Prin folosirea mai multor tipuri de reprezentări, dar și modele diferite, a fost posibilă o evaluare mai complexă, care a condus la realizarea unor comparații și, ulterior, la identificarea celor mai bune performanțe.

Pentru a antrena modelele, au fost utilizate trei tipuri diferite de reprezentări ale semnalelor sub forma unor imagini, cum ar fi melspectrogramele sau bazate pe Coeficienții Mel-Cepstrali sau Transformata Wavelet. Au fost alese aceste tipuri de reprezentări pentru a îmbogăți diversitatea descrierilor spectrale ale semnalelor, astfel oferind diferite caracteristici pe baza cărora se va realiza procesul de învățare. Partea de antrenare a constat și ea în împărțirea în două ramuri și anume utilizarea unei rețele neuronale convoluționale și a unei versiuni modificate a algoritmului CoAtNet, scopul fiind de a compara o abordare clasică cu un model mai modern și avansat. Structura scenariilor a influențat semnificativ rezultatele experimentale, prin faptul ca a oferit mai multe perspective asupra diferențelor în ceea ce privește curbele de învățare, capacitatea de a generaliza, dar și valorile obținute pentru metrici precum acuratețea, precizia sau senzitivitatea.

Rezultatele finale indică că cea mai bună performanță a fost obținută în cazul scenariului utilizării algoritmului CoAtNet și a melspectrogramelor ca suport pentru antrenare, având o acuratețe de 87,61%. Deși antrenarea CNN pe baza melspectrogramelor a obținut rezultate favorabile, acuratețea din acest caz este puțin mai scăzută față de scenariul precedent și anume 84,86%. Atât pentru CoAtNet, cât și pentru CNN, antrenarea folosind MFCC sau Transformata Wavelet oferă performanțe mai scăzute, ceea ce demonstrează că aceste reprezentări nu sunt suficient de interpretabile pentru a rezolva cerințele proiectului.

Proiectul a fost finalizat prin integrarea sistemului într-o aplicație web, pentru a oferi o interfață accesibilă prin care pot fi testate modelele. Aplicația permite încărcarea unui fișier audio ce conține apăsările tastelor, pentru care vor fi generate predicțiile individual. Cu ajutorul acesteia au fost îndeplinite două aspecte și anume, evidențierea capabilităților semnalului de a fi utilizat în cadrul aplicațiilor și oferirea unei imagini de ansamblu asupra predicțiilor.

În procesul de dezvoltare al acestui proiect, au existat și câteva dificultăți care pot fi atribuite atât etapei de documentare necesară realizării proiectului, aceasta fiind temelia de la care a fost posiblă o dezvoltare practică ulterioară, dar și etapei de implementare în care au apărut problemele reale.

Prima dificultate întâmpinată a fost în cadrul generării reprezentărilor pentru semnalele acustice, care a constat în asigurarea unei organizări corespunzătoare în directoare etichetate în timpul salvării locale pentru a evita apariția confuziilor între clase. Orice astfel de eroare ar fi putut influența negativ procesul de învățare, ducând la rezultate incorecte. Din punct de vedere al etapei de antrenare , un aspect problematic a fost optimizarea modelelor, atât prin intermediul arhitecturilor, cât și al hiperparametrilor, fiind necesare mai multe teste pentru a obține cea mai bună performanță. De asemenea, așa cum a fost menționat și în capitolul 4, o problemă persistentă a fost apariția fenomenului de supraînvățare, pentru care au fost implementate măsuri de regularizare, cum ar fi setarea parametrului „weight-decay” sau integrarea straturilor de dezactivare a neuronilor. În partea de generare a predicțiilor au existat probleme în ceea ce privește compatibilitatea reprezentărilor generate în această etapă cu cele utilizate în timpul antrenării modelelor. Pentru a asigura o predicție corectă, este necesar ca aceste reprezentări să fie obținute în mod identic.

Pe lângă aceste dificultăți, sistemul propus vine și cu o serie de limitări care pot impacta aplicabilitatea în anumite domenii sau acuratețea în alte contexte față de cele prezentate în lucrare.

Principala limitare a proiectului se regăsește la nivelul setului de date, care reduce robustețea sistemului din mai multe perspective. Un prim aspect care indică acest lucru constă în faptul că baza de date conține exclusiv apăsări ale tastelor corespunzătoare literelor din alfabetul englez, excluzând cifrele precum și diferite taste cu funcții, precum „shift” sau „enter”. În același timp, gradul redus de diversitate al înregistrărilor, care au fost realizate în condiții ideal, fără introducerea zgomotului sau modificarea echipamentelor utilizate, limitează capacitatea sistemului de a funcționa în scenarii reale. Un posibil progres legat de acest aspect este dezvoltarea bazei de date astfel încât să conțină exemple care reflectă mai bine un scenariu real și anume prin introducerea zgomotului, înregistrarea unor stiluri variate de dactilografiere sau utilizarea unor tastaturi care redau sunete diferite. De asemenea, dimensiunea redusă a setului de date limitează capacitatea modelului de a învăța trăsături generale, iar din acest motiv, o extindere a acestuia ar contribui direct la creșterea performanței algoritmilor.

Odată cu dezvoltarea setului de date, o posibilă continuare a lucrării este și integrarea blocurilor de „self-attention” cu cele convoluționale, urmând arhitectura inițială a algoritmului CoAtNet. Prin această metodă crește atât complexitatea modelului, dar și capacitatea acestuia de a surprinde relațiile prezente în cadrul datelor.

O altă limitare se referă la faptul că acuratețea predicțiilor realizate de sistem este influențată de utilizarea unei liste predefinite de cuvinte pentru a corecta secvențele de litere rezultate inițial. Prin această metodă este facilitată reconstrucția cuvintelor din limbajul natural, însă introduce o limitare în ceea ce privește flexibilitatea sistemului, astfel reducând aria sa de aplicabilitate în domenii precum cele legate de securitatea informațională. Pentru a adresa această limitare, este necesară eliminarea dependenței dintre lista de cuvinte și predicții prin îmbunătățirea performanței generale a modelului, astfel încăt să fie capabil de a recunoaște corect și secvențe aleatorii.

Proiectul propus poate fi continuat prin dezvoltarea unui sistem capabil să analizeze și să recunoască tiparul de tastare al utilizatorului, prin extragerea unor trăsături unice și specifice, precum ritmul apăsării tastelor, pauzele dintre litere sau forța cu care sunt apăsate tastele. O asemenea abordare ar presupune nu doar identificarea conținutului, ci și autentificarea utilizatorului, astfel oferind oportunitatea de utilizare a sistemului în mecanisme de control al accesului în structuri de informații.

O altă posibilă extindere a lucrării constă în integrarea proiectului cu un sistem geometric de localizare, realizat cu ajutorul tehnicilor de triangulație, cu scopul de a asocia fiecărei litere atât un sunet, cât și o poziție pe tastatură. Îmbinarea celor două abordări de recunoaștere a tastelor permite realizare unor distincții mai clare între taste care emit sunete similare, dar care sunt situate în zone diferite ale tastaturii, ceea ce duce la scăderea probabilității apariției predicțiilor incorecte.

# *Bibliografie*

[1] D. Asonov and R. Agrawal, “Keyboard acoustic emanations,” in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, Berkeley, CA, USA: IEEE, 2004, pp. 3–11. doi: 10.1109/SECPRI.2004.1301311.

[2] Mubarek Mohammed, “Acoustic Side Channel Attack on Keyboard,” 2024, doi: 10.13140/RG.2.2.29394.98246.

[3] Z. Kh. Abdul and A. K. Al-Talabani, “Mel Frequency Cepstral Coefficient and its Applications: A Review,” *IEEE Access*, vol. 10, pp. 122136–122158, 2022, doi: 10.1109/ACCESS.2022.3223444.

[4] L. Zhuang, F. Zhou, and J. D. Tygar, “Keyboard acoustic emanations revisited,” in *Proceedings of the 12th ACM conference on Computer and communications security*, Alexandria VA USA: ACM, Nov. 2005, pp. 373–382. doi: 10.1145/1102120.1102169.

[5] Y. Berger, A. Wool, and A. Yeredor, “Dictionary attacks using keyboard acoustic emanations,” in *Proceedings of the 13th ACM conference on Computer and communications security*, Alexandria Virginia USA: ACM, Oct. 2006, pp. 245–254. doi: 10.1145/1180405.1180436.

[6] T. Halevi and N. Saxena, “Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios,” *Int. J. Inf. Secur.*, vol. 14, no. 5, pp. 443–456, Oct. 2015, doi: 10.1007/s10207-014-0264-7.

[7] J. Harrison, E. Toreini, and M. Mehrnezhad, “A Practical Deep Learning-Based Acoustic Side Channel Attack on Keyboards,” 2023, doi: 10.48550/ARXIV.2308.01074.

[8] S. A. Anand and N. Saxena, “Keyboard Emanations in Remote Voice Calls: Password Leakage and Noise(less) Masking Defenses,” in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, Tempe AZ USA: ACM, Mar. 2018, pp. 103–110. doi: 10.1145/3176258.3176341.

[9] A. Compagno, M. Conti, D. Lain, and G. Tsudik, “Don’t Skype &amp; Type! Acoustic Eavesdropping in Voice-Over-IP,” 2016, *arXiv*. doi: 10.48550/ARXIV.1609.09359.

[10] Z. Martinasek, V. Clupek, and K. Trasy, “Acoustic attack on keyboard using spectrogram and neural network,” in *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, Prague, Czech Republic: IEEE, Jul. 2015, pp. 637–641. doi: 10.1109/TSP.2015.7296341.

[11] S. Turkle, *Alone together: why we expect more from technology and less from each other*. New York: Basic books, 2011.

[12] “How artificial intelligence is transforming the world,” Brookings. Accessed: Apr. 12, 2025. [Online]. Available: https://www.brookings.edu/articles/how-artificial-intelligence-is-transforming-the-world/

[13] I. H. Sarker, “Machine Learning: Algorithms, Real-World Applications and Research Directions,” *SN Comput. Sci.*, vol. 2, no. 3, p. 160, May 2021, doi: 10.1007/s42979-021-00592-x.

[14] D.-T. Mai, D.-C. Nguyen, and T.-N. Pham, “A Keystroke Audio-based Password Prediction Attack using Deep Learning,” in *2024 International Conference on Advanced Technologies for Communications (ATC)*, Ho Chi Minh City, Vietnam: IEEE, Oct. 2024, pp. 774–778. doi: 10.1109/ATC63255.2024.10908124.

[15] P. K., “Acoustic Wave,” in *Acoustic Waves - From Microdevices to Helioseismology*, M. G. Beghi, Ed., InTech, 2011. doi: 10.5772/17765.

[16] K. Lashkari, B. Friedlander, J. Abel, and B. McQuiston, “Classification of transient signals (acoustic signals),” in *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, New York, NY, USA: IEEE, 1988, pp. 2689–2692. doi: 10.1109/ICASSP.1988.197204.

[17] D. K. Staff, “All the Parts of a Mechanical Keyboard Explained,” Das Keyboard Mechanical Keyboard Blog. Accessed: Apr. 05, 2025. [Online]. Available: https://www.daskeyboard.com/blog/parts-of-a-mechanical-keyboard/

[18] M. S. Ahmad, “Deep Learning 101: Lesson 23: The Basics of Audio Signal Processing with FFT,” Medium. Accessed: Apr. 06, 2025. [Online]. Available: https://muneebsa.medium.com/deep-learning-101-lesson-23-the-basics-of-audio-signal-processing-with-fft-ffef65689c1d

[19] “Understanding the Hanning Window: A Practical Guide for Beginners,” Wray Castle. Accessed: Apr. 07, 2025. [Online]. Available: https://wraycastle.com/blogs/knowledge-base/hanning-window

[20] Y. A. Yousef, V. Cosman, and M. Rey, “Time Frequency Analyses of stationary and non-stationary signals using MATLAB functions and Toolboxes),” Jul. 17, 2024, *In Review*. doi: 10.21203/rs.3.rs-4749352/v1.

[21] L. Roberts, “Understanding the Mel Spectrogram,” Analytics Vidhya. Accessed: Apr. 07, 2025. [Online]. Available: https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53

[22] mlearnere, “Learning from Audio: The Mel Scale, Mel Spectrograms, and Mel Frequency Cepstral Coefficients,” TDS Archive. Accessed: Apr. 07, 2025. [Online]. Available: https://medium.com/data-science/learning-from-audio-the-mel-scale-mel-spectrograms-and-mel-frequency-cepstral-coefficients-f5752b6324a8

[23] “Audio Comparison Using MFCC and DTW,” TestDevLab Blog. Accessed: Apr. 09, 2025. [Online]. Available: https://www.testdevlab.com/blog/audio-comparison-using-mfcc-and-dtw

[24] H. Fayek, “Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What’s In-Between,” Haytham Fayek. Accessed: Apr. 09, 2025. [Online]. Available: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

[25] G. Strang, “The Discrete Cosine Transform,” *SIAM Rev.*, vol. 41, no. 1, pp. 135–147, Jan. 1999, doi: 10.1137/S0036144598336745.

[26] “Fourier vs. Wavelet Transformations,” Built In. Accessed: Apr. 10, 2025. [Online]. Available: https://builtin.com/data-science/wavelet-transform

[27] T. Konstantinovsky, “Wavelet Transform: A Practical Approach to Time-Frequency Analysis,” The Pythoneers. Accessed: Apr. 10, 2025. [Online]. Available: https://medium.com/pythoneers/wavelet-transform-a-practical-approach-to-time-frequency-analysis-662bdadeb08b

[28] B. Russel and J. Han, “Jean Morlet and the continuous wavelet transform.” CREWES Res., 2016.

[29] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[30] “What is deep learning? | McKinsey.” Accessed: Apr. 13, 2025. [Online]. Available: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-deep-learning

[31] D. M. Dobrea, *Tehnici de inteligenţă computaţională: aplicaţii în electronică şi biomedicină*. Iaşi: Performantica, 2018.

[32] M. A. Nielsen, “Neural Networks and Deep Learning,” 2015, Accessed: Apr. 13, 2025. [Online]. Available: http://neuralnetworksanddeeplearning.com

[33] “Activation Functions in Neural Networks [12 Types & Use Cases].” Accessed: Apr. 13, 2025. [Online]. Available: https://www.v7labs.com/blog/neural-networks-activation-functions

[34] R. Qamar and B. Ali Zardari, “Artificial Neural Networks: An Overview,” *Mesopotamian J. Comput. Sci.*, pp. 130–139, Aug. 2023, doi: 10.58496/MJCSC/2023/015.

[35] “Backpropagation in Neural Network | GeeksforGeeks.” Accessed: Apr. 18, 2025. [Online]. Available: https://www.geeksforgeeks.org/backpropagation-in-neural-network/

[36] “What is Backpropagation? | IBM.” Accessed: Apr. 18, 2025. [Online]. Available: https://www.ibm.com/think/topics/backpropagation

[37] “What is Gradient Descent? | IBM.” Accessed: Apr. 18, 2025. [Online]. Available: https://www.ibm.com/think/topics/gradient-descent

[38] “Recent advances and applications of deep learning methods in materials science | npj Computational Materials.” Accessed: Apr. 18, 2025. [Online]. Available: https://www.nature.com/articles/s41524-022-00734-6

[39] K. O’Shea and R. Nash, “An Introduction to Convolutional Neural Networks,” Dec. 02, 2015, *arXiv*: arXiv:1511.08458. doi: 10.48550/arXiv.1511.08458.

[40] “What are Convolutional Neural Networks? | IBM.” Accessed: Apr. 18, 2025. [Online]. Available: https://www.ibm.com/think/topics/convolutional-neural-networks

[41] “Fully Connected Layer vs. Convolutional Layer: Explained | Built In.” Accessed: Apr. 18, 2025. [Online]. Available: https://builtin.com/machine-learning/fully-connected-layer

[42] Z. Dai, H. Liu, Q. V. Le, and M. Tan, “CoAtNet: Marrying Convolution and Attention for All Data Sizes,” Sep. 15, 2021, *arXiv*: arXiv:2106.04803. doi: 10.48550/arXiv.2106.04803.

[43] A. Vaswani *et al.*, “Attention Is All You Need,” Aug. 02, 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.

[44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” Mar. 21, 2019, *arXiv*: arXiv:1801.04381. doi: 10.48550/arXiv.1801.04381.

[45] Johri, Prashant & Khatri, Sunil Kumar & Al-Taani, Ahmad & Sabharwal, Munish & Suvanov, Shakhzod & Chauhan, Avneesh., “Natural Language Processing: History, Evolution, Application, and Future Work,” 2021. doi: 10.1007/978-981-15-9712-1\_31.

[46] “What Is NLP (Natural Language Processing)? | IBM.” Accessed: May 05, 2025. [Online]. Available: https://www.ibm.com/think/topics/natural-language-processing

[47] “What is a confusion matrix? | IBM.” Accessed: Apr. 22, 2025. [Online]. Available: https://www.ibm.com/think/topics/confusion-matrix

[48] Schlosser, Tobias & Friedrich, Michael & Meyer, Trixy & Kowerko, Danny., “A Consolidated Overview of Evaluation and Performance Metrics for Machine Learning and Computer Vision.,” 2024.

[49] Cutting, Vineesh & Stephen, Nehemiah, “A Review on using Python as a Preferred Programming Language for Beginners,” vol. 8, pp. 4258–4263, 2021.

[50] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” 2019, *arXiv*. doi: 10.48550/ARXIV.1912.01703.

[51] “What is PyTorch? | IBM.” Accessed: Apr. 23, 2025. [Online]. Available: https://www.ibm.com/think/topics/pytorch

[52] “PyTorch Grows as the Dominant Open Source Framework for AI and ML: 2024 Year in Review,” PyTorch. Accessed: Apr. 23, 2025. [Online]. Available: https://pytorch.org/blog/2024-year-in-review/

[53] B. McFee *et al.*, “librosa: Audio and Music Signal Analysis in Python,” presented at the Python in Science Conference, Austin, Texas, 2015, pp. 18–24. doi: 10.25080/Majora-7b98e3ed-003.

[54] Dayanithi, “Streamlit in 2 Mins,” Data And Beyond. Accessed: May 17, 2025. [Online]. Available: https://medium.com/data-and-beyond/streamlit-d357935b9c

[55] J. Wu, *chinhsuanwu/coatnet-pytorch*. (Jun. 03, 2025). Python. Accessed: Jun. 04, 2025. [Online]. Available: https://github.com/chinhsuanwu/coatnet-pytorch

[56] “FrequencyWords/content/2018/ro/ro\_50k.txt at master · hermitdave/FrequencyWords,” GitHub. Accessed: May 27, 2025. [Online]. Available: https://github.com/hermitdave/FrequencyWords/blob/master/content/2018/ro/ro\_50k.txt

# *Anexe*

Anexa 1 – features.py

|  |
| --- |
| import os  import numpy as np  import librosa  import librosa.display  import matplotlib.pyplot as plt  from noisereduce import reduce\_noise  import soundfile as sf  import pywt  from PIL import Image  input\_folder = "E:\\AN4\\licenta\\val\_preprocesare\\Raw\_Data"  base\_output\_folders = {      "waveforms": "E:\\AN4\\licenta\\val\_preprocesare\\waveforms",      "segmented\_audio": "E:\\AN4\\licenta\\val\_preprocesare\\segmented\_audio",      "fft\_representations": "E:\\AN4\\licenta\\val\_preprocesare\\fft\_representations",      "mel\_spectrograms": "E:\\AN4\\licenta\\val\_preprocesare\\mel\_spectrograms",      "mfcc\_spectrograms": "E:\\AN4\\licenta\\val\_preprocesare\\mfcc\_spectrograms",      "wavelet\_spectrograms": "E:\\AN4\\licenta\\val\_preprocesare\\wavelet\_spectrograms"  }  for category, path in base\_output\_folders.items():      os.makedirs(path, exist\_ok=True)  def wavelet\_transform(segment, sr):      scales = np.arange(1, 64)      wavelet = 'morl'      coefficients, \_ = pywt.cwt(segment, scales, wavelet, 1.0 / sr)      return coefficients  for file in os.listdir(input\_folder):      if file.endswith(".wav"):          letter = os.path.splitext(file)[0]          audio\_path = os.path.join(input\_folder, file)          y, sr = librosa.load(audio\_path, sr=None)          y = y / np.max(np.abs(y))  y\_denoised = reduce\_noise(y=y, sr=sr)          onset\_frames = librosa.onset.onset\_detect(y=y\_denoised, sr=sr, delta=0.2, pre\_max=10, post\_max=10, units="time")  min\_time\_gap = 0.3          filtered\_onsets = [onset\_frames[0]]          for onset in onset\_frames[1:]:              if onset - filtered\_onsets[-1] > min\_time\_gap:                  filtered\_onsets.append(onset)          fixed\_length = 0.3          fixed\_samples = int(fixed\_length \* sr)          if len(filtered\_onsets) > 2:              filtered\_onsets = filtered\_onsets[:-1]          for category, base\_path in base\_output\_folders.items():              letter\_path = os.path.join(base\_path, letter)              os.makedirs(letter\_path, exist\_ok=True)          for idx in range(len(filtered\_onsets)-1):              start = int(filtered\_onsets[idx] \* sr)              end = int(filtered\_onsets[idx + 1] \* sr)              segment = y[start:end]              if len(segment) < fixed\_samples:                  segment = np.pad(segment, (0, fixed\_samples - len(segment)), mode='constant')              else:                  segment = segment[:fixed\_samples]              segment\_name = f"{letter}\_keystroke\_{idx + 1}"              sf.write(os.path.join(base\_output\_folders["segmented\_audio"], letter, f"{segment\_name}.wav"), segment, sr)  plt.figure(figsize=(10, 3))              librosa.display.waveshow(segment, sr=sr, alpha=0.8)              plt.xlabel("Time")              plt.ylabel("Amplitude")              plt.tight\_layout()              plt.savefig(os.path.join(base\_output\_folders["waveforms"], letter, f"{segment\_name}.png"))              plt.close()              fft\_values = np.fft.fft(segment)              fft\_magnitude = np.abs(fft\_values)              fft\_frequencies = np.fft.fftfreq(len(segment), d=1/sr)  plt.figure(figsize=(10, 4))              plt.plot(fft\_frequencies[:len(fft\_frequencies)//2], fft\_magnitude[:len(fft\_magnitude)//2])              plt.xlabel("Frequency")              plt.ylabel("Magnitude")              plt.tight\_layout()              plt.savefig(os.path.join(base\_output\_folders["fft\_representations"], letter, f"{segment\_name}.png"))              plt.close()                mel\_spectrogram = librosa.feature.melspectrogram(y=segment, sr=sr, n\_fft=2048, hop\_length=512, n\_mels=128)              mel\_spectrogram\_db = librosa.power\_to\_db(mel\_spectrogram, ref=np.max)  plt.figure(figsize=(10, 3))              librosa.display.specshow(mel\_spectrogram\_db, sr=sr, hop\_length=512, x\_axis = "time", y\_axis='mel')              plt.tight\_layout()              output\_path = os.path.join(base\_output\_folders["mel\_spectrograms"], letter, f"{segment\_name}.png")              plt.savefig(output\_path, transparent=False, facecolor="white")              plt.close()              img = Image.open(output\_path).convert("RGB")              img.save(output\_path)              mfccs = librosa.feature.mfcc(y=segment, sr=sr, n\_mfcc=13, n\_fft=2048, hop\_length=512)              mfccs = mfccs[1:, :]  plt.figure(figsize=(10, 3))              librosa.display.specshow(mfccs, sr=sr, hop\_length=512, x\_axis ='time')              plt.tight\_layout()              output\_path=os.path.join(base\_output\_folders["mfcc\_spectrograms"], letter, f"{segment\_name}.png")              plt.savefig(output\_path, transparent=False, facecolor="white")              plt.close()              img = Image.open(output\_path).convert("RGB")              img.save(output\_path)                wavelet\_coeffs = wavelet\_transform(segment, sr)  plt.figure(figsize=(10, 3))              plt.imshow(np.abs(wavelet\_coeffs), aspect='auto', extent=[0, fixed\_length, 1, 64])              plt.xlabel("Time")              plt.ylabel("Scales")              plt.tight\_layout()              output\_path=os.path.join(base\_output\_folders["wavelet\_spectrograms"], letter, f"{segment\_name}.png")              plt.savefig(output\_path, transparent=False, facecolor="white")              plt.close()              img = Image.open(output\_path).convert("RGB")              img.save(output\_path) |

Anexa 2 – cnn.py

|  |
| --- |
| import torch  import torch.nn as nn  import torch.optim as optim  from torch.utils.data import ConcatDataset, random\_split, DataLoader  from torchvision import datasets, transforms  from timeit import default\_timer as timer  from plot import plot\_loss  from plot import plot\_accuracy  from conf\_matrix import conf\_mat  import matplotlib.pyplot as plt  from sklearn.metrics import classification\_report  class CNN(nn.Module):      def \_\_init\_\_(self):          super().\_\_init\_\_()          self.conv1 = nn.Sequential(              nn.Conv2d(                  in\_channels=3,                  out\_channels=16,                  kernel\_size=3,                  stride=1,                  padding=1              ),              nn.BatchNorm2d(16),              nn.ReLU(),              nn.MaxPool2d(kernel\_size=2)          )          self.conv2 = nn.Sequential(              nn.Conv2d(                  in\_channels=16,                  out\_channels=32,                  kernel\_size=3,                  stride=1,                  padding=1              ),              nn.BatchNorm2d(32),              nn.ReLU(),              nn.MaxPool2d(kernel\_size=2),              nn.Dropout(p=0.1)          )          self.conv3 = nn.Sequential(              nn.Conv2d(                  in\_channels=32,                  out\_channels=64,                  kernel\_size=3,                  stride=1,                  padding=1              ),              nn.BatchNorm2d(64),              nn.ReLU(),              nn.MaxPool2d(kernel\_size=2),              nn.Dropout(p=0.1)          )          self.conv4 = nn.Sequential(              nn.Conv2d(                  in\_channels=64,                  out\_channels=128,                  kernel\_size=3,                  stride=1,                  padding=1              ),              nn.ReLU(),              nn.BatchNorm2d(128),              nn.MaxPool2d(kernel\_size=2),              nn.Dropout(p=0.2)          )          self.conv5 = nn.Sequential(              nn.Conv2d(                  in\_channels=128,                  out\_channels=256,                  kernel\_size=3,                  stride=1,                  padding=1              ),              nn.ReLU(),              nn.BatchNorm2d(256),              nn.MaxPool2d(kernel\_size=2),              nn.Dropout(p=0.2)            )          self.conv6 = nn.Sequential(              nn.Conv2d(                  in\_channels=256,                  out\_channels=512,                  kernel\_size=3,                  stride=1,                  padding=1              ),              nn.BatchNorm2d(512),              nn.ReLU(),              nn.MaxPool2d(kernel\_size=2),              nn.Dropout(p=0.3)          )          self.conv7 = nn.Sequential(              nn.Conv2d(                  in\_channels=512,                  out\_channels=1024,                  kernel\_size=3,                  stride=1,                  padding=1              ),              nn.BatchNorm2d(1024),              nn.ReLU(),              nn.MaxPool2d(kernel\_size=2),              nn.Dropout(p=0.4)          )          self.global\_pool = nn.AdaptiveAvgPool2d((1, 1))          self.flatten = nn.Flatten()          self.fc1 = nn.Linear(1024, 512)          self.relu = nn.ReLU()          self.dropout = nn.Dropout(p=0.5)          self.fc2 = nn.Linear(512, 26)        def forward(self, input\_data):          x = self.conv1(input\_data)          x = self.conv2(x)          x = self.conv3(x)          x = self.conv4(x)          x = self.conv5(x)          x = self.conv6(x)          x = self.conv7(x)          x = self.global\_pool(x)          x = self.flatten(x)          x = self.fc1(x)          x = self.relu(x)          x = self.dropout(x)          x = self.fc2(x)            return x  def split(train\_dir, val\_dir, test\_dir, batch\_size=8, train\_ratio=0.7, val\_ratio=0.15, test\_ratio=0.15):      transform = transforms.Compose([          transforms.Resize((224, 224)),          transforms.ToTensor()      ])      train\_dataset = datasets.ImageFolder(train\_dir, transform=transform)      val\_dataset = datasets.ImageFolder(val\_dir, transform=transform)      test\_dataset = datasets.ImageFolder(test\_dir, transform=transform)      full\_dataset = ConcatDataset([train\_dataset, val\_dataset, test\_dataset])      total\_size = len(full\_dataset)      train\_size = int(train\_ratio \* total\_size)      val\_size = int(val\_ratio \* total\_size)      test\_size = total\_size - train\_size - val\_size      new\_train, new\_val, new\_test = random\_split(full\_dataset, [train\_size, val\_size, test\_size])      train\_loader = DataLoader(new\_train, batch\_size=batch\_size, shuffle=True)      val\_loader = DataLoader(new\_val, batch\_size=batch\_size, shuffle=False)      test\_loader = DataLoader(new\_test, batch\_size=batch\_size, shuffle=False)      class\_names = train\_dataset.classes      return train\_loader, val\_loader, test\_loader, class\_names  def train(model, train\_loader, loss\_fn, optimizer, device):      model.train()      train\_loss, correct = 0, 0      total = 0      for batch, (X, y) in enumerate(train\_loader):          X, y = X.to(device), y.to(device)          y\_pred = model(X)          loss = loss\_fn(y\_pred, y)          optimizer.zero\_grad()          loss.backward()          optimizer.step()          train\_loss += loss.item()          correct += (y\_pred.argmax(dim=1) == y).sum().item()          total += y.size(0)          accuracy = 100 \* correct / total      return train\_loss / len(train\_loader), accuracy  def validate(model, val\_loader, loss\_fn, device):      model.eval()      val\_loss, correct = 0, 0      total = 0      with torch.inference\_mode():          for X, y in val\_loader:              X, y = X.to(device), y.to(device)              y\_pred = model(X)              val\_loss += loss\_fn(y\_pred, y).item()              correct += (y\_pred.argmax(dim=1) == y).sum().item()              total += y.size(0)      accuracy = 100 \* correct / total      return val\_loss / len(val\_loader), accuracy  def test(model, test\_loader, loss\_fn, device):      model.eval()      test\_loss, correct = 0, 0      total = 0      model.eval()      preds\_list = []      labels = []      with torch.inference\_mode():          for X, y in test\_loader:              X, y = X.to(device), y.to(device)              y\_pred = model(X)              test\_loss += loss\_fn(y\_pred, y).item()              correct += (y\_pred.argmax(dim=1) == y).sum().item()              total += y.size(0)                preds = y\_pred.argmax(dim=1)              preds\_list.extend(preds.tolist())              labels.extend(y.tolist())      accuracy = 100 \* correct / total      return test\_loss / len(test\_loader), accuracy, torch.tensor(preds\_list), torch.tensor(labels)  def print\_report(y, y\_pred, class\_names, filename):      report = classification\_report(y.cpu(), y\_pred.cpu(), target\_names=class\_names, digits=3,  zero\_division=0)        print("Classification Report:")      print(report)        with open(filename, "w") as f:          f.write(report)        print(f"Report saved to {filename}")  def train\_and\_evaluate(model, train\_loader, val\_loader, test\_loader, loss\_fn, optimizer, device, epochs=30):      total\_training\_time = 0      best\_val\_acc = 0.0      train\_loss\_values = []      val\_loss\_values = []      train\_acc\_values = []      val\_acc\_values = []      epoch\_count = []      for epoch in range(epochs):          print(f"\nEpoch {epoch+1}/{epochs}")            epoch\_start\_time = timer()            train\_loss, train\_acc = train(model, train\_loader, loss\_fn, optimizer, device)          val\_loss, val\_acc = validate(model, val\_loader, loss\_fn, device)          print(f"Train Loss: {train\_loss:.4f} | Train accuracy: {train\_acc:.2f}% | Val Loss: {val\_loss:.4f} | Val Accuracy: {val\_acc:.2f}%")            if val\_acc > best\_val\_acc:              best\_val\_acc = val\_acc              torch.save(model.state\_dict(), "best\_model\_wave\_cnn.pth")            epoch\_end\_time = timer()          epoch\_duration = epoch\_end\_time - epoch\_start\_time          total\_training\_time += epoch\_duration          print(f"Epoch {epoch+1} took {epoch\_duration:.2f} seconds | Total training time so far: {total\_training\_time:.2} seconds")            epoch\_count.append(epoch)          train\_loss\_values.append(train\_loss)          val\_loss\_values.append(val\_loss)          train\_acc\_values.append(train\_acc)          val\_acc\_values.append(val\_acc)        print(f"\nBest Validation Accuracy: {best\_val\_acc:.2f}%")      plot\_loss(epoch\_count, train\_loss\_values, val\_loss\_values)      plot\_accuracy(epoch\_count, train\_acc\_values, val\_acc\_values)        test\_loss, test\_acc, preds\_list, labels = test(model, test\_loader, loss\_fn, device)      print(f"Test Loss: {test\_loss:.4f} | Test Accuracy: {test\_acc:.2f}%")        return preds\_list, labels  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  num\_classes = 26  model = CNN().to(device)  loss\_fn = nn.CrossEntropyLoss()  optimizer = optim.Adam(model.parameters(), lr=0.0001, weight\_decay=0.0001)  train\_loader, val\_loader, test\_loader, class\_names = split(      train\_dir="E:\\AN4\\licenta\\dataset\_wave\_oficial\\train",      val\_dir="E:\\AN4\\licenta\\dataset\_wave\_oficial\\val",      test\_dir="E:\\AN4\\licenta\\dataset\_wave\_oficial\\test",      batch\_size=8  )  y\_pred, labels = train\_and\_evaluate(model, train\_loader, val\_loader, test\_loader, loss\_fn, optimizer, device, epochs=50)  fig, ax = conf\_mat(y\_pred, labels, num\_classes, class\_names, figsize=(10,7))  plt.show()  print\_report(labels, y\_pred, class\_names, filename="classification\_report\_wave\_cnn.txt") |

Anexa 3 – coatnet.py

|  |
| --- |
| import torch  import torch.nn as nn  import torch.optim as optim  from timeit import default\_timer as timer  from plot import plot\_loss  from plot import plot\_accuracy  from conf\_matrix import conf\_mat  import matplotlib.pyplot as plt  from sklearn.metrics import classification\_report  from torch.utils.data import ConcatDataset, random\_split, DataLoader  from torchvision import datasets, transforms  def conv\_3x3\_bn, inp, oup, image\_size, downsample=False):      stride = 1 if not downsample else 2      return nn.Sequential(          nn.Conv2d(inp, oup, kernel\_size=3, stride=stride, padding=1, bias=False),          nn.BatchNorm2d(oup),          nn.GELU()      )  class MBConv(nn.Module):      def \_\_init\_\_(self, inp, oup, image\_size, downsample=False, expansion=4):          super().\_\_init\_\_()          self.downsample = downsample          stride = 1 if not downsample else 2          hidden\_dim = int(inp \* expansion)          if self.downsample:              self.pool = nn.MaxPool2d(3, 2, 1)              self.proj = nn.Conv2d(inp, oup, kernel\_size=1, stride=1, padding=0, bias=False)          if expansion == 1:              self.conv = nn.Sequential(                  nn.Conv2d(hidden\_dim, hidden\_dim, kernel\_size=3, stride=stride, padding=1, groups=hidden\_dim, bias=False),                  nn.BatchNorm2d(hidden\_dim),                  nn.GELU(),                  nn.Conv2d(hidden\_dim, oup, kernel\_size=1, stride=1, padding=0, bias=False),                  nn.BatchNorm2d(oup),              )          else:              self.conv = nn.Sequential(                  nn.Conv2d(inp, hidden\_dim, kernel\_size=1, stride=stride, padding=0, bias=False),                  nn.BatchNorm2d(hidden\_dim),                  nn.GELU(),                  nn.Conv2d(hidden\_dim, hidden\_dim, kernel\_size=3, stride=1, padding=1, groups=hidden\_dim, bias=False),                  nn.BatchNorm2d(hidden\_dim),                  nn.GELU(),                  nn.Conv2d(hidden\_dim, oup, kernel\_size=1, stride=1, padding=0, bias=False),                  nn.BatchNorm2d(oup),              )        def forward(self, x):          if self.downsample:              return self.proj(self.pool(x)) + self.conv(x)          else:              return x + self.conv(x)  class CoAtNet(nn.Module):      def \_\_init\_\_(self, image\_size, in\_channels, num\_blocks, channels, num\_classes=26):          super().\_\_init\_\_()          ih, iw = image\_size          self.s0 = self.\_make\_layer(conv\_3x3\_bn, in\_channels, channels[0], num\_blocks[0], (ih // 2, iw // 2))          self.s1 = self.\_make\_layer(MBConv, channels[0], channels[1], num\_blocks[1], (ih // 4, iw // 4))          self.s2 = self.\_make\_layer(MBConv, channels[1], channels[2], num\_blocks[2], (ih // 8, iw // 8))          self.s3 = self.\_make\_layer(MBConv, channels[2], channels[3], num\_blocks[3], (ih // 16, iw // 16))          self.s4 = self.\_make\_layer(MBConv, channels[3], channels[4], num\_blocks[4], (ih // 32, iw // 32))          self.pool = nn.AdaptiveAvgPool2d((1, 1))          self.dropout = nn.Dropout(p=0.5)          self.fc = nn.Linear(channels[-1], num\_classes, bias=False)      def forward(self, x):          x = self.s0(x)          x = self.s1(x)          x = self.s2(x)          x = self.s3(x)          x = self.s4(x)          x = self.pool(x).view(x.size(0), -1)          x = self.dropout(x)          x = self.fc(x)          return x      def \_make\_layer(self, block, inp, oup, depth, image\_size):          layers = nn.ModuleList([])          for i in range(depth):              if i == 0:                  layers.append(block(inp, oup, image\_size, downsample=True))              else:                  layers.append(block(oup, oup, image\_size))          return nn.Sequential(\*layers)  def coatnet\_0():      #num\_blocks = [2, 2, 3, 5, 2]      num\_blocks = [2, 2, 2, 2, 1]      channels = [64, 96, 192, 384, 768]      return CoAtNet((224, 224), 3, num\_blocks, channels, num\_classes=26) |

Anexa 4 – predict.py

|  |
| --- |
| import librosa  import librosa.display  from PIL import Image  import torchvision.transforms as transforms  import os  import numpy as np  import pywt  import matplotlib.pyplot as plt  from noisereduce import reduce\_noise  import uuid  import torch  import torch.nn as nn  from itertools import product  def wavelet\_transform(segment, sr):      scales = np.arange(1, 64)      wavelet = 'morl'      coefficients, \_ = pywt.cwt(segment, scales, wavelet, 1.0 / sr)      return coefficients  def generate\_image(segment, sr, rep\_type, temp\_img\_path):      if rep\_type == "Melspectrogram":          mel\_spec = librosa.feature.melspectrogram(y=segment, sr=sr, n\_fft=2048, hop\_length=512, n\_mels=128)          mel\_spec\_db = librosa.power\_to\_db(mel\_spec, ref=np.max)          plt.figure(figsize=(10, 3))          librosa.display.specshow(mel\_spec\_db, sr=sr, hop\_length=512)          plt.savefig(temp\_img\_path, transparent=False, facecolor="white")          plt.close()      elif rep\_type == "MFCC":          mfccs = librosa.feature.mfcc(y=segment, sr=sr, n\_mfcc=13, n\_fft=2048, hop\_length=512)          mfccs = mfccs[1:, :]          plt.figure(figsize=(10, 3))          librosa.display.specshow(mfccs, sr=sr, hop\_length=512)          plt.savefig(temp\_img\_path, transparent=False, facecolor="white")          plt.close()      elif rep\_type == "Wavelet":          coeffs = wavelet\_transform(segment, sr)            plt.figure(figsize=(10, 3))          plt.imshow(np.abs(coeffs), aspect='auto', extent=[0, 0.3, 1, 64])          plt.savefig(temp\_img\_path, transparent=False, facecolor="white")          plt.close()      img = Image.open(temp\_img\_path).convert("RGB")      os.remove(temp\_img\_path)      return img  def predict(model, audio\_path, rep\_type="Melspectrogram", class\_names=None, device='cpu', word\_list=None):      model.eval()      y, sr = librosa.load(audio\_path, sr=None)      y = y / np.max(np.abs(y))      y\_denoised = reduce\_noise(y=y, sr=sr)      onset\_frames = librosa.onset.onset\_detect(y=y\_denoised, sr=sr, delta=0.2, pre\_max=10, post\_max=10, units="time")      audio\_duration = librosa.get\_duration(y=y, sr=sr)      if len(onset\_frames) == 0 or onset\_frames[-1] < audio\_duration - 0.1:          onset\_frames = np.append(onset\_frames, audio\_duration)      min\_time\_gap = 0.3      filtered\_onsets = [onset\_frames[0]]      for onset in onset\_frames[1:]:          if onset - filtered\_onsets[-1] > min\_time\_gap:              filtered\_onsets.append(onset)      fixed\_length = 0.3      fixed\_samples = int(fixed\_length \* sr)      all\_top\_probs = []      all\_top\_indices = []      predicted\_letters = []      top\_7\_letters\_per\_segment = []      transform = transforms.Compose([          transforms.Resize((224, 224)),          transforms.ToTensor()      ])      for idx in range(len(filtered\_onsets) - 1):          start = int(filtered\_onsets[idx] \* sr)          end = int(filtered\_onsets[idx + 1] \* sr)          segment = y[start:end]          if len(segment) < fixed\_samples:              segment = np.pad(segment, (0, fixed\_samples - len(segment)), mode='constant')          else:              segment = segment[:fixed\_samples]          temp\_img\_path = f"temp\_{uuid.uuid4().hex}.png"          img = generate\_image(segment, sr, rep\_type, temp\_img\_path)          input\_tensor = transform(img).unsqueeze(0).to(device)          img.save(f"letter\_{idx}.png")            with torch.inference\_mode():              output = model(input\_tensor)              probs = torch.softmax(output, dim=1).squeeze()              top\_probs, top\_indices = torch.topk(probs, k=min(7, len(probs)))              all\_top\_probs.append(probs.cpu().numpy())              all\_top\_indices.append(top\_indices.cpu().numpy())              pred\_idx = top\_indices[0].item()              predicted\_letter = class\_names[pred\_idx] if class\_names else str(pred\_idx)              predicted\_letters.append(predicted\_letter)              top\_letters = [class\_names[i] for i in top\_indices]              top\_7\_letters\_per\_segment.append(top\_letters)              print(f"Segment {idx}: Predicted Letter = {predicted\_letter}, Top-7 = {top\_letters}")      print("\nInitial Prediction:", ''.join(predicted\_letters))      candidates = list(product(\*all\_top\_indices))      words = [''.join(class\_names[i] for i in word) for word in candidates]      word\_scores = [np.prod([probs[i] for probs, i in zip(all\_top\_probs, word)]) for word in candidates]      if word\_list:          filtered = [(w, s) for w, s in zip(words, word\_scores) if w in word\_list]          filtered.sort(key=lambda x: x[1], reverse=True)          corrected = filtered[0][0] if filtered else words[np.argmax(word\_scores)]          print("Top candidate matches:")          for cand, score in filtered[:5]:              print(f"{cand}")          all\_top\_predictions = [w for w, s in filtered]      else:          corrected = words[np.argmax(word\_scores)]          all\_top\_predictions = words      print("Corrected Word:", corrected)      return predicted\_letters, corrected, all\_top\_predictions, top\_7\_letters\_per\_segment  with open("ro\_words.txt", "r", encoding="utf-8") as f:      custom\_word\_list = set(line.strip().split()[0].lower() for line in f if line.strip())  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  model = CNN().to(device)  model.load\_state\_dict(torch.load("E:\\AN4\\licenta\\best\_model\_mel\_cnn.pth", map\_location=device))  class\_names = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']  audio\_file = "E:\\AN4\\licenta\\parola\_3.wav"  predicted\_word = predict(model, audio\_file, rep\_type="Melspectrogram", class\_names=class\_names, device=device, word\_list=custom\_word\_list)  print("Predicted Word:", predicted\_word) |

Anexa 5 – app.py

|  |
| --- |
| import streamlit as st  import uuid  from predict import predict, generate\_image  import torch  import torch.nn as nn  st.set\_page\_config(page\_title="Keystroke Recognition", layout="wide")  st.markdown("""      <style>          body {              background-color: #f8f9fa;          }          .title {              font-size: 42px;              font-weight: bold;              color: #800020;              text-align: center;              margin-top: 0px;              margin-bottom: 20px;          }          .section-header {              font-size: 26px;              color: #A70D2A;              margin-top: 30px;          }          .prediction-box {              background-color: #ffffff;              padding: 15px;              border-radius: 10px;              border: 2px solid #A70D2A;              margin-top: 5px;              font-size: 18px;          }          .upload-section, .settings-section {              background-color: #A70D2A;              padding: 1px;              margin-bottom: 5px;          }      </style>  """, unsafe\_allow\_html=True)  st.markdown('<div class="title">Platformă de predicții pentru semnale acustice emise prin dactilografiere</div>', unsafe\_allow\_html=True)  with open("ro\_words.txt", "r", encoding="utf-8") as f:      word\_list = set(line.strip().split()[0].lower() for line in f if line.strip())  class\_names = [      'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',      'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'  ]  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  st.markdown('<div class="upload-section">', unsafe\_allow\_html=True)  st.markdown("### Încarcă fișierul WAV")  uploaded\_file = st.file\_uploader("Încarcă fișierul audio:", type=["wav"])  st.markdown('</div>', unsafe\_allow\_html=True)  st.markdown('<div class="settings-section">', unsafe\_allow\_html=True)  st.markdown("### Setările pentru model și reprezentare")  model\_choice = st.selectbox("Alege modelul:", ["CoAtNet", "CNN"])  rep\_type = st.selectbox("Alege tipul de reprezentare:", ["Melspectrogram", "MFCC", "Wavelet"])  st.markdown('</div>', unsafe\_allow\_html=True)  if st.button("Realizează predicția"):      if uploaded\_file is None:          st.warning("Este necesară încărcarea unui fișier audio înaintea realizării predicției.")      else:          temp\_audio\_path = f"temp\_audio\_{uuid.uuid4().hex}.wav"          with open(temp\_audio\_path, "wb") as f:              f.write(uploaded\_file.read())          st.audio(temp\_audio\_path)          if model\_choice == "CoAtNet":              model = coatnet\_0().to(device)              if rep\_type == "Melspectrogram":                  model\_path = "E:/AN4/licenta/best\_model\_mel\_coatnet.pth"              elif rep\_type == "MFCC":                  model\_path = "E:/AN4/licenta/best\_model\_mfcc\_coatnet.pth"              elif rep\_type == "Wavelet":                  model\_path = "E:/AN4/licenta/best\_model\_wave\_coatnet.pth"              else:                  st.error("Nu există acest tip de reprezentare pentru algoritmul CoAtNet")                  st.stop()          elif model\_choice == "CNN":              model = CNN().to(device)              if rep\_type == "Melspectrogram":                  model\_path = "E:/AN4/licenta/best\_model\_mel\_cnn.pth"              elif rep\_type == "MFCC":                  model\_path = "E:/AN4/licenta/best\_model\_mfcc\_cnn.pth"              elif rep\_type == "Wavelet":                  model\_path = "E:/AN4/licenta/best\_model\_wave\_cnn.pth"              else:                  st.error("Nu există acest tip de reprezentare pentru modelul CNN.")                  st.stop()          else:              st.error("Model nu există.")              st.stop()          try:              model.load\_state\_dict(torch.load(model\_path, map\_location=device))          except Exception as e:              st.error(f"Eroare de încărcare a modelului: {e}")              st.stop()          predicted\_letters, corrected, all\_top\_predictions, top\_7\_letters\_per\_segment = predict(              model=model,              audio\_path=temp\_audio\_path,              rep\_type=rep\_type,              class\_names=class\_names,              device=device,              word\_list=word\_list          )            st.markdown('<div class="section-header"> Predicția inițială</div>', unsafe\_allow\_html=True)          st.markdown(f'<div class="prediction-box">{"".join(predicted\_letters)}</div>', unsafe\_allow\_html=True)          st.markdown('<div class="section-header"> Primele 7 predicții pentru fiecare literă</div>', unsafe\_allow\_html=True)          for i, top7 in enumerate(top\_7\_letters\_per\_segment):              st.markdown(f'<div class="prediction-box"><b>Litera {i+1}:</b> {", ".join(top7)}</div>', unsafe\_allow\_html=True)          st.markdown('<div class="section-header"> Top cuvinte alternative</div>', unsafe\_allow\_html=True)          st.markdown(f'<div class="prediction-box">{", ".join(all\_top\_predictions[:5])}</div>', unsafe\_allow\_html=True)          st.markdown('<div class="section-header"> Cuvântul final corectat</div>', unsafe\_allow\_html=True)          st.markdown(f'<div class="prediction-box">{corrected}</div>', unsafe\_allow\_html=True) |

Anexa 6 – plot.py, matrix.py

|  |
| --- |
| import matplotlib.pyplot as plt  def plot\_loss(epoch\_count, train\_loss\_values, val\_loss\_values):      plt.figure(figsize=(8, 6))      plt.plot(epoch\_count, train\_loss\_values, label="Train loss")      plt.plot(epoch\_count, val\_loss\_values, label="Validation loss")      plt.title("Loss Curves")      plt.xlabel("Epochs")      plt.ylabel("Loss")      plt.legend()      plt.show()    def plot\_accuracy(epoch\_count, train\_acc\_values, val\_acc\_values):        plt.figure(figsize=(8, 6))      plt.plot(epoch\_count, train\_acc\_values, label="Train Accuracy")      plt.plot(epoch\_count, val\_acc\_values, label="Validation Accuracy")      plt.title("Accuracy Curves")      plt.xlabel("Epochs")      plt.ylabel("Accuracy (%)")      plt.legend()      plt.show()  from torchmetrics import ConfusionMatrix  from mlxtend.plotting import plot\_confusion\_matrix  def conf\_mat(y\_pred, labels, num\_classes, class\_names, figsize=(10, 7)):      confmat = ConfusionMatrix(num\_classes = len(class\_names), task='multiclass')      confmat\_tensor = confmat(preds=y\_pred, target=labels)        fig, ax = plot\_confusion\_matrix(          conf\_mat=confmat\_tensor.numpy(),          class\_names=class\_names,          figsize=figsize      )        return fig, ax |

1. Mecanism utilizat în domeniul învățării automate pentru a surprinde dependențe și relații în cadrul secvențelor de intrare [↑](#footnote-ref-1)
2. O curbă în formă de clopot, similară unei distribuții normale, utilizată pentru a modula o altă funcție [↑](#footnote-ref-2)
3. Software al cărui cod sursă este accesibil utilizatorilor pentru a fi viuzalizat sau modificat [↑](#footnote-ref-3)
4. O structură predefinită care oferă baza pentru dezvoltarea aplicațiilor software [↑](#footnote-ref-4)
5. Un model de bază utilizat pentru reprezentarea și afișarea culorilor, în care fiecare literă se referă la un canal separat (Red, Green, Blue), uneori adăugându-se și litera A care indică transparența imaginii [↑](#footnote-ref-5)
6. Dezactivează un anumit procent de neuroni pentru a preveni fenomentul de supraînvățare [↑](#footnote-ref-6)
7. Ajustează valorile de intrare pentru a se încadra în anumite intervale [↑](#footnote-ref-7)
8. Transformă datele în format unidimensional [↑](#footnote-ref-8)